C++ семестр

1. Повтор С

Знакомство:
Какие языки знаете? кто был на С, на Java ,
В любом случае повторить основы С

История С++
С разработан Dennis Ritchie в1969-1973 at AT&T Bell Labs.
С++ — компилируемый статически типизированный язык программирования общего назначения. С++ это инкремент от С
Популярные компиляторы GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder
Синтаксис С++ унаследован от языка С. Одним из принципов разработки было сохранение совместимости с С
Наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования
Язык возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьёрн Страуструп(Bjarne Stroustrup) придумал ряд усовершенствований к языку С под собственные нужды.
ISO стандарты:

1998	**ISO/IEC 14882:1998**	**C++98**
2003	ISO/IEC 14882:2003	C++03
2007	ISO/IEC TR 19768:2007	C++TR1
2011	**ISO/IEC 14882:2011**	**C++11**
2014	N3797 (working draft C++14)	C++14
TBD	N4687	c++20

Среда:
Code::Blocks — свободная кроссплатформенная среда разработки. Code::Blocks написана на C++ и использует библиотеку wxWidgets.
Code::Block 20
gcc 4.x
allegro 5.2.2[для графики]
Windows 10(https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/)

Можно так же и Visual Studio Community Edition(вроде только 15-ое и 16-ое занятие нельзя будет сделать в Visual Studio Community Edition, так как все настройки под codeblock написаны, но если поиграться с Visual Studio — то там тоже можно сделать, просто нету инструкций):
1. установить Visual Studio Community Edition — проверял на 2019
2. установить NuGet
Extensions→Manage Extensions найти NUGET Package Version Updater
3. Начать новый проект: New Console Application
4. добавить Allegro(для первого занятие это не нужно)
правый клик на проект → Managet NuGet packages
установить Allegro (проверял на 5.2.x)
Если не может найти, то надо добавить https://api.nuget.org/v3/index.json в Package Source(справа кнопочка шестерёнка)
5. подключить необходимые модули
правый клик на проект → Properties → Configuration Properties → Allegro 5 → Add-ons
выбрать Primitives Addon = Yes

6. добавить необходимые файлы в проект
правый клик на проект Add→Existing Items и выбрать AllegroUtil.cpp/hpp(они должны быть скопированы в папку где лежит main фаил проекта(например ConsoleApplication1.cpp)
7. затем Build→Rebuild

Можно использовать и свой редактор, тогда вручную компилить\линковать
CmakeLists.txt взят от CLion

```
cmake_minimum_required(VERSION 3.19)
project(untitled C)
set(CMAKE_C_STANDARD 99)
add_executable(untitled main.c AllegroUtils.cpp)
target_link_libraries (untitled allegro allegro_primitives )
```

Так же я попробывал в Visual Studio Code с тулчайном от CodeBlocks20
Предполагаю что C:\CodeBlocks20 скопирован
в корне проекта

```
.vscode\c_cpp_properties.json
{
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**",
                "C:/CodeBlocks20/MinGW/include"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "windowsSdkVersion": "10.0.19041.0",
            "compilerPath": "C:/CodeBlocks20/MinGW/bin/gcc.exe",
            "cStandard": "c17",
            "cppStandard": "c++17",
            "intelliSenseMode": "windows-gcc-x64"
        }
    ],
    "version": 4
}
.vscode\launch.json
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?
linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "g++.exe - Build and debug active file",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${fileDirname}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
```

```
            "miDebuggerPath": "C:\\CodeBlocks20\\MinGW\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: g++.exe build active file ver(1)"
        }
    ]
}
.vscode\settings.json
{
    "files.associations": {
        "iostream": "cpp"
    }
}
.vscode\tasks.json
{
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++.exe build active file",
            "command": "C:\\CodeBlocks20\\MinGW\\bin\\g++.exe",
            "args": [
                "-g",
                "${workspaceFolder}\\*.cpp",
                "-I",
                "C:\\CodeBlocks20\\MinGW\\include",
                "-L",
                "C:\\CodeBlocks20\\MinGW\\lib",
                "-lliballegro",
                "-lliballegro_primitives",
                "-o",
                "${workspaceFolder}\\main.exe"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: codeblocks g++.exe"
        },
        {
            "type": "cppbuild",
            "label": "C/C++: g++.exe build active file ver(1)",
            "command": "C:\\CodeBlocks20\\MinGW\\bin\\g++.exe",
            "args": [
                "-g",
                "${workspaceFolder}\\*.cpp",
                "-I",
                "C:\\CodeBlocks20\\MinGW\\include",
                "-L",
                "C:\\CodeBlocks20\\MinGW\\lib",
                "-lliballegro",
                "-lliballegro_primitives",
                "-o",
                "${workspaceFolder}\\main.exe"
            ],
            "options": {
```

```
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "detail": "Task generated by Debugger."
        }
    ],
    "version": "2.0.0"
}
```

запуск через cmd:
PATH=%PATH%;C:\CodeBlocks20\MinGW\bin
code .
ctrl+shift+b – билд
ctrl+F5 - запуск
F5 – запуск через отладчик

1. Объявление переменных
`void testVariables()`
В любом месте, не обязательно в начале блока
Лучше сразу же инициализировать каким-то значением, выражением.
2.0 - принято для дробных. float - только когда проблема с памятью, или для графики, иначе double
(int)f - округление.

2. Операторы
```
001 #include <iostream> \\ это пока магия
002
003 using namespace std; \\ это тоже
```
`void testOperators()`
Пример с выводом, вводом значений.
>=, <=,==, !=, !, &&, ||, &,|
<<(insertion operator), >>(extraction operator) - очень по простому, зависит от места в кода, в C++ это не всегда битовые сдвиги.
/, %, *, +, -, (,)

в C++ есть "C Library", то есть <cmath>, <cstdio>, <cstdlib>, <cstring> и другие

3. Циклы(+ условия)
`void testLoop()`
Циклы for\while\do while + continue, break
if-else if - else
switch ( ) case : default: break;

4. Функции
`void testFunctions()`
Обратить внимание на форвард декларацию. Фунция в функции невозможна в C\C++

5. Массивы
`void testArrays()`
Массивы либо как *, либо как [], константы, и константные массивы

~~6. ссылки (совсем не обязательно, если уж время осталось)~~

7. Прикольно получилось, если рассказать struct с ф-циями внутри и к этому привизать практические занятия.

8. Расмотреть примеры Luch\pas2cpp\ (совсем не обязательно, если уж время осталось)

Ресурсы:
http://www.sgi.com/tech/stl/table_of_contents.html
http://yosefk.com/c++fqa/
http://www.cplusplus.com/forum/lounge/28407/
http://www.parashift.com/c++-faq/
Различия, ясно и коротко : http://psi-logic.narod.ru/pro/diffs.htm (Программирование - 10 отличий C от C++.html)
http://wiki.allegro.cc/index.php?title=Allegro_5_API_Tutorials
http://www.learncpp.com

Для детей:
C : http://www.freebsd.org/cgi/man.cgi
C++ : http://cplusplus.com/reference и http://cppreference.com
если не помогло, то www.google.com

Задания на практике:

Условия:
1. Перевод температур, реализовать меню выбора.
C- цельсий, F – Фаренгейт, K- Кельвин, Реомюра(R)
R = 0.8 * C; F = 1.8 * C; K = 273.15 + C
Ввод : темпереатура в в цельсиях(C) и пункт меню
Вывод :
Перевод :

1. **Фаренгейт**
2. **Кельвин**
3. **Отмена**

Сделать структуру Converter(double t; double toF(), double toC())

2. Максимум и минимум среди 3ёх чисел

3. Квадратное уравнения. Вводятся a,b,c - найти корни
axx + bx + c = 0 , x1,x2 = (-b +/- sqrt(bb-4ac) / 2a
struct Roots{n, x1, x2}; struct SquareEq { a,b,c; Roots solve()}

Циклы:
4. Вычислить N-е число Фибоначи и сумму первых N чисел Фибоначи.
6. $\frac{1}{1\cdot 2\cdot 3}+\frac{1}{2\cdot 3\cdot 4}+\frac{1}{3\cdot 4\cdot 5}+\frac{1}{4\cdot 5\cdot 6}+\cdots$
7. $y=\sin^2 x$ $y=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{2^{2n-1}*x^{2n}}{(2n)!}$
8. Совершенное число́ — натуральное число, равное сумме всех своих младших делителей (т. е. всех делителей, отличных от само́го числа).

Первое совершенное число — 6 (1 + 2 + 3 = 6), следующее — 28 (1 + 2 + 4 + 7 + 14 = 28). По мере того как натуральные числа возрастают, совершенные числа встречаются всё реже. Третье совершенное число — 496, четвёртое — 8 128, пятое — 33 550 336, шестое — 8 589 869 056.

Массивы:
9. Перевернуть массив задом наперед
10. Задача Иосифа (n воинов, стоящих по кругу, и убивают каждого m-го, массив 0\1 есть или нет человека)
11. Вычислить 2^N, N=1, 2, … 2000. Примечание: 2^2000 состоит из 603 цифр. Хранить число как массив цифр. struct Power2{ char data[1000]; void calc(int n); void print()}

Более-менее интересное и простое.
12) Игра «Жизнь».
Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам: пустая (мёртвая) клетка ровно с тремя живыми клетками-соседями оживает; если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае (если соседок меньше двух или больше трёх(по диагонали)) клетка умирает (от «одиночества» или от «перенаселённости»).
Начальное положение хранится в файле.
#include "windows.h"
COORD c = {0,0};
SetConsoleCursorPosition(GetStdHandle(STD_OUPUT_HANDLE), c);
Sleep(1000);
13) Золотая горка(спустится с горы собрав как можно больше золота)
14) В написанном выражении ((((1 ? 2) ? 3) ? 4) ? 5) ? 6 вместо каждого знака ? вставить знак одной из 4 арифметических операций +, -, *, : так, чтобы результат вычислений равнялся заданному целому числу n (при делении дробная часть отбрасывается).
20 = (((((1 + 2) + 3) * 4) * 5) / 6)
20 = (((((1 * 2) + 3) + 4) + 5) + 6)
20 = (((((1 * 2) * 3) * 4) * 5) / 6)

15) Битва роботов. комп vs хуман. Робот имеет здоровье, умеет бить и умеет защищаться. На каждый ход можно ударить в одну из точек(голова, корпус, ноги) и можно защитить одну из точек. Выиграл тот кто выжил. Сделать через структуры
+ оружие 1d10 2 хода (топор), 1d3 1 ход(кинжал)

```
001 #include <iostream>
003 using namespace std;
004
005 void testVariables()
006 {
007     int x;
008     x = 10;
010     unsigned int y = 10;
012     int z = x + y;
014     float f = 2.0;
015     double d = 4.0;
017     int a = (int)f;
018     double b = 2 * y + 3 * x;
020 }
021
022 #include <cmath>
023
024 void testOperators()
025 {
026     int x = 7 / 3;
027     int y = 7 % 3;
028     cout << x << " " << y << endl;
030     double x1;
031     cin >> x1;
032     cout << sqrt( x1 ) << endl;
033     cout << sin( x1 ) << endl;
034 }
035
036 void testLoop()
037 {
038     int sum = 0;
039     for( int i = 0; i < 10; ++i )
040     {
041         sum += i;
042     }
043     cout << sum << endl;
045     int x;
046     do
047     {
048       cout << "positive num=" << endl;
049       cin >> x;
050     }
051     while ( x <= 0 );
```

```
053     int d = 0;
054     while( x > 0 )
055     {
056         d += x % 10;
057         x /= 10;
058     }
059     cout << "count = " << d << endl;
060
061     for( int i = 0; ; ++i )
062     {
063         if ( ( i % 2 ) == 0 )
065             continue;
067         if ( i > 5 )
069             break;
071         cout << i << " ";
072     }
073     cout << endl;
074 }
075
076 void f1();
077 int f2( int x, int y );
079 void testFunctions()
080 {
081     f1();
082     cout << f2( 1, 2 ) << endl;
083 }
085 void f1()
086 {
087     cout << "f1" << endl;
088 }
090 int f2( int x, int y )
091 {
092     return ( x + y ) / 2;
093 }
094
095 void setSquares( int a[], int n )
096 {
097     for( int i = 0; i < n; ++i )
098     {
099        a[ i ] = ( i + 1 ) * ( i + 1 );
100     }
101 }
```

```
103 void print( const int *a, int n )
104 {
105     for( int i = 0; i < n; ++i )
106     {
107         cout << a[i] << " ";
108     }
109     cout << endl;
110 }
112 const int SIZE = 10;
113 void testArrays()
114 {
115     int a[SIZE];
116     setSquares( a, SIZE );
117     print(a, sizeof(a)/sizeof(a[0]));
118 }
120 struct Robot
121 {
122     int n = 0;
124     void touch()
125     {
126         n++;
127         cout << "You touched a robot "
<< n << " times!" << endl;
128     }
130     void sum(int a, int b)
131     {
132         cout << a + b << endl;
133     }
134 };
136 void testStruct()
137 {
138     Robot robot;
139     robot.touch();
140     robot.sum(1, 2);
141     robot.touch();
142 }
```

C : http://www.freebsd.org/cgi/man.cgi
C++ : http://cplusplus.com/reference и http://cppreference.com если не помогло, то www.google.com
ej.uz/vladm-prog cpp\bundle codeblocks20

```c
int bitSum(unsigned n) {
    int s=0;
    do
        s += n & 1;
    while(n >>= 1);
    return s;
}
```

```pascal
function BitSum(n: integer):integer;
var s: integer;
begin
    s := 0;
    while n <> 0 do begin
        Inc(s, n and 1);
        n := n shr 1;
    end;
    BitSum := s;
end;
```

```c
int nod(int n, int m) { // recursive
    if (n == m)
        return n;
    if (n > m)
        return nod(n % m, n);
    return nod(n, m % n);
}

int nod(int n, int m) {
    while (n != m)
        if (n < m)
            m = m % n;
        else
            n = n % m;
    return n;
}
```

```pascal
function Nod(a, b: longint): longint; {recursive}
begin
  if (a > b) then
    Nod := Nod(a mod b, b)
  else
    if (a < b) then
      Nod := Nod(a, b mod a)
    else
      Nod := a;
end;

function Nod(a, b: longint): longint;
begin
  while (a <> b) do
    if (a < b) then
      b := b mod a
    else
      a := a mod b;
  Nod:=a;
end;
```

```c
#include <iostream>

int arrayMin(int n, const int data[]) {
    int min = data[0];
    for(int i = 1; i < n; ++i)
        if(data[i] < min)
            min = data[i];
    return min;
}
```

```pascal
function ArrayMin(n: integer; data: array of
integer):integer;
var i, min: integer;
begin
    min := data[0];
    for i := 1 to n - 1 do
        if data[i] < min then
            min := data[i];
    ArrayMin := min;
end;
```

2. Классы, Allegro

A class is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.

Представление об объектах

Большинство программ на C++ отражают реальные существующие объекты. В известном смысле объекты представляют собой сущности, например автомобиль, собаку, часы и т. д. Обычно объект имеет несколько атрибутов и операций, которые программа должна выполнять над этими атрибутами. Например, в случае часов свойства могут включать текущее время и время будильника. Операции такого объекта могли бы содержать установку времени, установку будильника или выключение будильника. При объектно-ориентированном программировании ваши программы фокусируются на объектах и операциях над этими объектами.

1) void TestStruct()

В C++ структуры это классы. То есть могут содержать и методы. Как и другие переменные их надо инициализировать.

Item item = { 1.0, 2 }; так скомпилится, но это плохо, очень не понятно. Решение будет в следующем примере

Подчеркнуть что в конце объявления структуры или классы необходима ";" - компилятор иногда странные ошибки выдаёт.

2) void TestClass()

Дефолтный конструктор, он есть всегда если явно не определён другой конструктор. Его так же можно переопределить. Список инициализации. (пока что не напрягать с const & для аргументов и с copy-конструктором) . Семантика вызова коснтрукторов

Class x;

Class x(arg1, arg2);

Классы это структуры с защитой данных. Секции public, private.

Префиксы для приватных данных field_(от гугла https://google.github.io/styleguide/cppguide.html), _field(зарезервировано для библиотек), m_Field, mField …

Временные объекты.

Дефолтный конструктор(очень кратко, просто как описать, вызвать, и если нужны C массивы)

```
class A {
public:
      A() { ... };
}
A a, b[100]; // возможно надо будет проинициализировать
b[i].init();
```

3) void TestInheritance()

По сравнению с Java, в c++ нету общего родителя у всех классов.

~~public/protected/private - наследование .~~ Только о public наследовании.

Пока что-то виртуальные методы не трогать, на следующем занятие будет

Не только конструкторы могут иметь несколько версий, но и методы, например Move

Дальше с Аллегро, можно и до практики отложить.

4) Затем разобраться с Allegro(на практике все вместе)

New console application

Подключить либы(правый клик по проекту, build options->Linker settings)
C:\codeblocks\MinGW\lib\liballegro_primitives.dll.a
C:\codeblocks\MinGW\lib\liballegro.dll.a

Чтобы запускать из виндовса(не из ИДЕ)
В %PATH% прописать путь к DLL для allegro:
C:\codeblocks\mingw\bin;

Скопировать AllegroUtil.cpp/hpp и добавить в проект

4) Рассказать немного о строение программы: TestAllegro int main()
цикл:
- обработка событий(например нажатие клавишы)
- обработка изменний связанными со временем
- прорисовка

5) void draw1()
Основные элементы:

6) void draw2()
Рассказать что прорисовка на самом деле происходит в буффер. И только когда кадр полностью отрисован, он отсылается для отображения
Это делает команда - al_flip_display

7) void draw3() / void fps3()
Рассмотреть динамику.

Задание на практику:
ВНИМАНИЕ. сказать чтобы сохраняли программу где-то, чтобы потом можно было бы восстановить(нужно так как постепенно будем наварачивать программу)
На десктопе должен быть урл к хелпу Аллегро.
Запустить TestAllegro. Создать свой проект.
~~0) Цель : чтобы хоть что-то нарисовали~~
~~Нарисовать треугольник серпинского(размером близким к экрану)~~

Имя Rectangle конфликтует с Win GDI Api

Хелп по аллегро file:///C:/CodeBlocks20/MinGW/allegro/docs/html/refman/index_all.html

1) Цель : написать класс
Квадрат который бегает по экрану и отскакивает от стенок.
Square(x, y, side, dx, dy)

2) Внутри квадрата есть круг, который бегает внутри самого квадрата.
Иерархию наследования лучше не делать, слишком мало знают ещё. Можно обойтись включением.
Сделать классы Point(x, y), Square(center, side, velocity), SquareWithCircle, Circle(center, radius, velocity)
Пользователь вводит размер квадрата, размер круга в процентах от размера квадрата(50% - диаметр круга = 1/2 от стороны)

3) Есть много таких квадратов, отскакивают друг от друга.

Класс SquareScreenSaver
Пользователь вводит кол-во квадратов, всё остальное рандомно(размерности, расположение, цвет, скорость движения)

```
001 struct Item
002 {
003     double p; // price
004     int n; // count
005
006     void IncCount( int x )
007     {
008         n += x;
009     }
010
011     void Print()
012     {
013         cout << n << "x" << p << endl;
014     }
015 };
016
017 void TestStruct()
018 {
019     Item item;
020     item.Print(); // prints uninitialized data!!!
021     item.p = 10.0;
022     item.n = 2;
023     item.Print(); // 2x10
024     item.IncCount( 5 );
025     item.Print(); // 7x10
026 }
027
028 struct Point
029 {
030     double x;
031     double y;
032
033     Point( double x0, double y0 ) :
034         x( x0 ),
035         y( y0 )
036     {
037     }
038
039     void Add( Point other )
040     {
041         x += other.x;
042         y += other.y;
043     }
044
045     void Print()
046     {
047         cout << x << " " << y << endl;
048     }
049 };
050
051 class Line
052 {
053     public:
054         Line( double x1, double y1, double x2, double y2 ) :
055             p1_( x1, y1 ),
056             p2_( x2, y2 )
057         {
058         }
059         Line( Point p1, Point p2 ) :
060             p1_( p1.x, p1.y ),
061             p2_( p2.x, p2.y )
062         {
063         }
064
065         double GetLength()
066         {
067             return sqrt((p1_.x - p2_.x) * (p1_.x - p2_.x) +
068                         (p1_.y - p2_.y) * (p1_.y - p2_.y));
069         }
070
071         void Print()
072         {
073             p1_.Print();
074             p2_.Print();
075         }
076
077     private:
078         Point p1_;
079         Point p2_;
080 };
```

```
081
082 void TestClass()
083 {
084     Line l1( 1.0, 0.0, 10.0, 0.0 );
085     cout << l1.GetLength() << endl; // 9
086     Point p1( 2.0, 0.0 );
087     Point p2( 5.0, 0.0 );
088     Line l2( p1, p2 );
089     l2.Print(); // 2 0 \n 5 0
090     Line l3( Point( 1.0, 2.0 ), Point( 3.0, 4.0 ) );
091     l3.Print(); // 1 2 \n 3 4
092 }
093
094 class Figure
095 {
096     public:
097         Figure( Point p ) :
098             center_( p )
099         {
100         }
101
102     protected:
103         Point center_;
104 };
105
106 class Square : public Figure
107 {
108     public:
109         Square( Point center, double side ) :
110             Figure( center ),
111             side_( side )
112         {
113         }
114
115         void Print()
116         {
117             cout << side_ << " at " << center_.x << ", " <<
center_.y << endl;
118         }
119
120     protected:
121         double side_;
122 };
123
124 class DynamicSquare : public Square
125 {
126     public:
127         DynamicSquare( Point center, double side ) :
128             Square( center, side )
129         {
130         }
131
132         void Move( Point d )
133         {
134             center_.Add( d );
135         }
136         void Move( double dx, double dy )
137         {
138             center_.Add( Point( dx, dy ) );
139         }
140     private:
141 };
142
143 void TestInheritance()
144 {
145     Square s1( Point( 1.0, 2.0 ), 5.0 );
146     s1.Print(); // 5 at 1, 2
147     DynamicSquare s2( Point( 1.0, 1.0 ), 2.0 );
148     s2.Move( Point( 1.0, 2.0 ) );
149     s2.Print(); // 2 at 2, 3
150     s2.Move( 1.0, 2.0 );
151     s2.Print(); // 2 at 3, 5
152 }
```

```
01 #include <iostream>
02 #include "AllegroUtil.hpp"
03 #include <windows.h>
04 #include <cstdlib>
05
06 using namespace std;
07
08 const int FPS = 60;
09 const int SCREEN_W = 640;
10 const int SCREEN_H = 480;
11
12 int i = 0;
13 void draw1()
14 {
15     ++i;
16     cout << "frame " << i << endl;
17
18     al_clear_to_color( al_map_rgb( 0, 0, 0 ) );
19     al_put_pixel( 50, 50, al_map_rgb( 0, 255, 0 ) );
20     al_draw_line( 100, 100, 300, 200,
                     al_map_rgb( 255, 0, 0 ), 5 );
21     al_draw_triangle( 120, 120, 150, 120, 130, 150,
                         al_map_rgb( 255, 0, 0 ), 3 );
22     al_draw_filled_triangle( 120, 220, 150, 220, 130, 250,
                                al_map_rgb( 0, 255, 0 ) );
23     al_draw_rectangle( 300, 300, 350, 350,
                          al_map_rgb( 0, 255, 0 ), 1 );
24     al_draw_filled_rectangle( 350, 300, 400, 350,
                                 al_map_rgb( 0, 255, 255 ) );
25     al_draw_circle( 500, 400, 50, al_map_rgb(0, 255, 0), 3 );
26     al_draw_filled_circle(400,400,50, al_map_rgb(0, 255,0) );
27 }
28
29 void draw2()
30 {
31     for( int i = 0; i < 10; ++i )
32     {
33         al_draw_line( 100, 100, 200 + i * 10, 200,
                         al_map_rgb( 255, 0, 0 ), 2 );
34         al_flip_display();
35         Sleep( 1000 ); // one second sleep, from <windows.h>
36     }
37     ExitAllegro();
38 }
39
40 struct Circle
41 {
42     double x;
43     double y;
44     double dx;
45     double dy;
46     double r;
47     void Reset()
48     {
49         x = SCREEN_W / 2;
50         y = SCREEN_H / 2;
51         r = 10.0 + rand() % 100;
52         dx = 10.0 - rand() % 21;
53         dy = 10.0 - rand() % 21;
54     }
55 };
56
57 Circle circle;
58 void fps3()
59 {
60     circle.x += circle.dx;
61     circle.y += circle.dy;
62     if ( ( circle.x < 1.0 ) ||
63          ( circle.x > SCREEN_W ) ||
64          ( circle.y < 1.0 ) ||
65          ( circle.y > SCREEN_H ) )
66     {
67         circle.Reset();
68     }
69 }
70
71 void draw3()
72 {
73     al_clear_to_color( al_map_rgb( 0, 0, 0 ) );
74     al_draw_filled_circle( circle.x, circle.y, circle.r,
                              al_map_rgb( 0, 255, 0 ) );
75 }
```

```
76
77 int main(int argc, char **argv)
78 {
79     srand( time(0) );
80     if( !InitAllegro( SCREEN_W, SCREEN_H, FPS ) )
81     {
82         DestroyAllegro();
83         return 1;
84     }
85
86     //RunAllegro( 0, &draw1 );
87     //RunAllegro( 0, &draw2 );
88     circle.reset();
89     RunAllegro( &fps3, &draw3 );
90
91     DestroyAllegro();
92     // cin.get();
93     return 0;
94 }


Check compiler:
settings -> compiler -> GNU GCC Compiler
Tollchain Executable == "c:\CodeBlocks16\MinGW
Check debugger:
Settings -> debugger
Default: Executable path == c:\CodeBlocks16\MinGW\bin\gdb.exe
```

```
(right click on project)Build options:
1)Search directories -> Compiler
C:\codeblocks\MinGW\include
2)Linker Settings
C:\codeblocks\MinGW\lib\liballegro.dll.a
C:\codeblocks\MinGW\lib\liballegro_primitives.dll.a
3) если линковщик ругеается, то ещё
Search directories -> Linker
C:\codeblocks\MinGW\bin

Copy to root AllegroUtil.cpp/hpp and add files(right click on
project)

To run standalone windows need to find DLLs:
C:\codeblocks\MinGW\bin\allegro-5.0.dll
C:\codeblocks\MinGW\bin\allegro_primitives-5.0.dll
Add to %PATH% or copy to executable folder(..your_project\bin\
Debug)
Help: C:\codeblocks\MinGW\allegro\docs\html\refman\index_all.html

ВНИМАНИЕ! Имя Rectangle кофликтует с windows.h => использовать
что-то другое. Например, Rect

Google style guide:
https://google.github.io/styleguide/cppguide.html
```

3. Полиморфизм

Полиморфи́зм (от греч. πολὺ- — много, и μορφή — форма) в языках программирования — возможность объектов с одинаковой спецификацией иметь различную реализацию.
Виртуальный метод - метод класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения.

1. void TestVirtual()

Если ф-ция объявленена как виртуал в базовом классе, то она по умолчанию виртуальна в потомке. Но лучше явно это указывать. Рассмотреть построение класса.
В C++ указатель на базовый класс возможно присвоить указатель потомка(примерно так же как в Делфях, помните что все объекты в Делфи это на самом деле указатели(ссылки)).
То есть :

```
Child c;
Parent *p = &c;
p->M1() // доступен интерфейс Parent, но содержимое от Child.
```

Рассмотреть пример Caller1. Можно нарисовать дерево.
Спросить что выведет Caller2.

2. void TestMemory

Динамические объекты.
malloc/calloc -> new / версия для массивов new []
free -> delete / версия для массивов delete []
В случае с массивом оператор delete сам знает кол-во элеменотов в массиве. Поэтому его указывают при создание, кол-во это выражение, может быть не константой.
В C++ не принято использовать NULL, принято 0.
Перед delete можно не проверять на 0, так как delete ничего не сделает если ему передать нулевой указатель.

```
if ( !p ) // бессмысленное условие
{
      delete p;
}
```

хорошем тоном является обнуления укзателя вручную после delete(как и в C, этого не происходит автоматически)
В этом примере не совсем правильно, нету виртуального деструктора. но об этом в след. примере.

3. void TestDestructors()

Деструкторы
Вызывается когда освобождается объект(выходит из области видимости, или через delete)
Важность виртуальных деструкторов. Правило, если объект предполагает быть в качестве базового класса, то деструктор всегда должен быть виртуальным. Поэтому в базовом классе его надо объявить даже если он ничего не делает. Можно спросить что выведется на экране.
Важно что вызывать в конструкторе базового класса виртуальные методы нельзя(так как предки ещё не созданы), и вызвать в деструкторе виртуальные методы(так как предки уже удалены)

4. void TestReferences()

\\ Если мало времени то лучше на следующей лекции, 5-ый пример важнея
const ссылки пока что не трогать.
В C++ ссылки это как var-параметры из Паскаля. Синтаксис простой, если для указателя применятся *, для ссылок &.

Ссылки инициализируется только один раз в отличии от указателей. Для аргументов ф-ции компилятор это делает за вас. Это можно сделать в ручную.
Ссылки не могут иметь нулевой адресс(будет runtime-ошибка).
Ссылка хранится на стеке, так как же как и указатель. Размер ссылки равен размеру указателя.
В C++ принято использовать ссылки, где это удобно. Обращение к адресуемой переменной гораздо проще, чему через указатель. Можно считать что ссылка - это синоним какой-то другой переменной.
В делфи все объекты является ссылками, отличие от C++ что в делфи они могут быть равные nil, в c++ это привидёт к runtine-ошибки.

5. Рассмотреть пример TestAllegro
Метод Draw из базового класса посути является абстрактным, но об этом на следующем занятии. В примере нету виртуального деструктора.

Задание:
1) Сделать иерархию на подобии: Figure, Square, Circle. Летают и отскакивают от стенок.
2) Выделить список в отдельный класс (FigureList, SreenSaver...).
3) Добавить сталкивание(можно по простому, сравнить расстояние от центров. Для круга - это радиус, для квадрата - это половина стороны)
4) Добавить Figure -> MoveableFigure, SizeableFigure.
MoveableFigure->MoveableSquare, MoveableCircle
SizeableFigure ->(меняет своё размер, то есть если в MoveableFigure меняются координаты, то в SizeableFigure меняется коэфициент размера(для круга радиус, для квадрата сторона) SizeableCircle, SizeableSquare
Например, круг у которого меняется радису(уменьшается до 20% и затем увеличивается). Для начала можно о столкновениях не думать, но потом надо и это реализовать
5) Сделать FigureList класс потомком от Figure и заставить перемещатся так же как и остальные фигуры. Для этого нужно сделать чтобы FigureList имел свои границы(для простоты пускай это квадрат) и передовал их своим объектам и фигуры бы рисовались относительно границ FigureList.
5) Добавить прямоугольник который переползает(подтягивает свой край к другому краю)

```
006 class Parent
007 {
008 public:
009     void M1()
010     {
011         cout << "Parent::M1" << endl;
012     }
013     virtual void M2()
014     {
015         cout << "Parent::M2" << endl;
016     }
017     void M3()
018     {
019         cout << "Parent::M3" << endl;
020         M4();
021     }
022     virtual void M4()
023     {
024         cout << "Parent::M4" << endl;
025     }
026 };
027
028 class Child : public Parent
029 {
030 public:
031     void M1()
032     {
033         cout << "Child::M1" << endl;
034     }
035     virtual void M2()
036     {
037         cout << "Child::M2" << endl;
038     }
039     virtual void M4()
040     {
041         cout << "Child::M4" << endl;
042     }
043 };
044
045 void Caller1( Parent *p )
046 {
047     p->M1();
048     p->M2();
049 }
050
051 void Caller2( Parent *p )
052 {
053     p->M3();
054 }
055
056 void TestVirtual()
057 {
058     Parent p;
059     Caller1( &p ); // Parent::M1\nParent::M2
060     Child c;
061     Caller1( &c ); // Parent::M1\nChild::M2
062
063     Caller2( &p );
064     Caller2( &c );
065 }
066
067 typedef Parent * PParent;
068 void TestMemory()
069 {
070     {
071         int *p = new int;
072         *p = 100;
073         cout << *p << endl;
074         delete p;
075         p = 0;
076
077         Parent *base = new Child();
078         base->M1();
079         base->M2();
080         delete base;
081         base = 0;
082     }
083     {
084         int *p = new int[10];
085         for( int i = 0; i < 10; ++i )
086         {
087             p[i] = i * i;
```

```
088         }
089         delete[] p;
090         p = 0;
091
092         // Create using default constructor
093         Parent *bases = new Parent[5];
094         for( int i = 0; i < 5; ++i )
095         {
096             bases[i].M1();
097             bases[i].M2();
098         }
099         delete[] bases;
100         bases = 0;
101
102         // 5 pointers to Parent, no memory allocation
103         // Parent *bases2[5];
104         PParent bases2[5];
105         cout << "Virtual test" << endl;
106         for( int i = 0; i < 5; ++i )
107         {
108             if ( ( i % 2 ) == 0 )
109             {
110                 bases2[i] = new Parent();
111             }
112             else
113             {
114                 bases2[i] = new Child();
115             }
116             bases2[i]->M1();
117             bases2[i]->M2();
118         }
119
120         for( int i = 0; i < 5; ++i )
121         {
122             delete bases2[i];
123             bases2[i] = 0;
124         }
125     }
126 }
127
128 class ClassA
129 {
130 public:
131     ClassA()
132     {
133         cout << "ClassA::ClassA" << endl;
134     }
135     virtual ~ClassA()
136     {
137         cout << "ClassA::~ClassA" << endl;
138     }
139
140 };
141
142 class ClassB : public ClassA
143 {
144 public:
145     int *p;
146     ClassB() :
147         ClassA()
148     {
149         p = new int[10];
150         cout << "ClassB::ClassB, new 10 ints" << endl;
151     }
152     virtual ~ClassB()
153     {
154         delete[] p;
155         cout << "ClassB::~ClassB, free 10 ints" << endl;
156     }
157
158 };
159
160 void TestDestructors()
161 {
162     {
163         ClassA a;
164         ClassB b;
165     }
166     {
167         ClassA *base = new ClassB();
168         // If destructor is not virtual then
169         // ~ClassA will be called  and memory is not freed
```

```
170         delete base;
171     }
172 }
173
174 void ChangeIntPointer( int *x )
175 {
176     *x = 100;
177 }
178
179 void ChangeIntRef( int &x )
180 {
181     x = 200;
182 }
183
184 void Caller3( Parent &p )
185 {
186     p.M1();
187     p.M2();
188 }
189
190 void TestReferences()
191 {
192     int x;
193     ChangeIntPointer( &x );
194     cout << x << endl;
195     ChangeIntRef( x );
196     cout << x << endl;
197
198     int *pX = &x;
199     *pX = 300;
200     cout << x << " " << *pX << endl;
202     int &refX = x;
203     refX = 400;
204     cout << x << " " << refX << endl;
205
```

```
206     Parent p;
207     Child c;
208     Caller3( p );
209     Caller3( c );
210     Parent *p2 = new Child();
211     Caller3( *p2 );
212     delete p2;
213     p2 = 0;
214     //Caller3( *p2 ); // runtime error
215 }
```

```
009 const int FPS = 60;
010 const int SCREEN_W = 640;
011 const int SCREEN_H = 480;
012
013 class Figure
014 {
015 protected:
016     double x_;
017     double y_;
018     double dx_;
019     double dy_;
020
021 public:
022     Figure()
023     {
024         Reset();
025     }
026
027     void Reset()
028     {
029         x_ = rand() % SCREEN_W;
030         y_ = rand() % SCREEN_H;
031         dx_ = 10.0 - rand() % 21;
032         dy_ = 10.0 - rand() % 21;
033     }
034
035     virtual void Draw(){}
036
037     virtual void Move()
038     {
039         x_ += dx_;
040         y_ += dy_;
041         if ( ( x_ < 1.0 ) || ( x_ > SCREEN_W ) ||
043              ( y_ < 1.0 ) || ( y_ > SCREEN_H ) )
045         {
046             Reset();
047         }
048     };
049 };
050 typedef Figure * PFigure;
051
```

```
052 class Square : public Figure
053 {
054 protected:
055     double a_;
056 public:
057     Square( double a ) :
058         Figure(),
059         a_( a )
060     {
061     }
062     virtual void Draw()
063     {
064         double half = a_ / 2;
065         al_draw_filled_rectangle( x_ - half, y_ - half,
066             x_ + half, y_ + half, al_map_rgb( 255, 0, 0 ) );
067     }
068 };
069
070 class Circle : public Figure
071 {
072 protected:
073     double r_;
074     unsigned char color_;
075 public:
076     Circle( double r ) :
077         Figure(),
078         r_( r ),
079         color_( rand() % 256 )
080     {
081     }
082     virtual void Draw()
083     {
084         ++color_;
085         al_draw_filled_circle( x_, y_, r_, al_map_rgb( 0,
color_, 0 ) );
086     }
087 };
088
089 const int MAX = 100;
090 class ScreenSaver
091 {
```

```
092 private:
093     PFigure figures[MAX];
094     int size_;
095
096 public:
097     ScreenSaver() :
098         size_( 0 )
099     {
100         // Set to null all pointers
101         memset( figures, 0, sizeof( figures ) );
102     }
104     ~ScreenSaver()
105     {
106         for( int i = 0; i < size_; ++i )
107         {
108             delete figures[i];
109             figures[i] = 0;
110         }
111     }
112
113     void Draw()
114     {
115         al_clear_to_color( al_map_rgb( 0, 0, 0 ) );
116         for( int i = 0; i < size_; ++i )
117         {
118             figures[i]->Draw();
119         }
120     }
122     void Next()
123     {
124         for( int i = 0; i < size_; ++i )
125         {
126             figures[i]->Move();
127         }
128     }
129
130     void Add( Figure *f )
131     {
132         if ( size_ >= MAX )
133         {
134             return;
135         }
136         figures[ size_ ] = f;
137         ++size_;
138     }
139
140 };
141
142 ScreenSaver ss;
143
144 void fps()
145 {
146     ss.Next();
147 }
149 void draw()
150 {
151     ss.Draw();
152 }
153
154 int main(int argc, char **argv)
155 {
156     srand( time(0) );
157     if( !InitAllegro( SCREEN_W, SCREEN_H, FPS ) )
158     {
159         DestroyAllegro();
160         return 1;
161     }
162     for( int i = 0; i < 100; ++i )
163     {
164         if ( ( i % 2 ) == 0 )
165         {
166             ss.Add( new Circle( 10.0 + rand() % 30 ) );
167         }
168         else
169         {
170             ss.Add( new Square( 10.0 + rand() % 30 ) );
171         }
172     }
173     RunAllegro( &fps, &draw );
175     DestroyAllegro();
178     return 0;
179 }
```

4. Абстрактные методы, интерфейсы

Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона. В C++ нету чистых интерфейсов, в Java например есть. Интерфейсы C++ реализуются при помощи чисто виртуальных методов(абстрактных).

0. void TestReferences()
Рассказать если небыло рассказано:
const ссылки пока что не трогать.
В C++ ссылки это как var-параметры из Паскаля. Синтаксис простой, если для указателя применятся *, для ссылок &.
Ссылки инициализируется только один раз в отличии от указателей. Для аргументов ф-ции компилятор это делает за вас. Это можно сделать в ручную.
Ссылки не могут иметь нулевой адрес(будет runtime-ошибка).
Ссылка хранится на стеке, так как же как и указатель. Размер ссылки равен размеру указателя.
В C++ принято использовать ссылки, где это удобно. Обращение к адресуемой переменной гораздо проще, чем через указатель. Можно считать что ссылка - это синоним какой-то другой переменной.
В делфи все объекты является ссылками, отличие от C++ что в делфи они могут быть равные nil, в c++ это привидёт к runtine-ошибки. Ошибка будет при первом обращение(получение данных:вызов метода с this, или виртуальный, вывод на экран через разыменовать) по 0-ссылки.

1. void TestAbstract()
В базовом классе можно некоторые методы делать чисто виртуальными для того чтобы классы потомки предоставляли реализацию, иначе компилятор выдаст ошибку. Так же это делает базовый класс абстрактным(то есть если есть хотя бы один метод чисто виртуальным, либо в самом классе, либо унаследованный, то класс является абстрактным). Невозможно создать экземляр абстрактного класса.

2. void TestInterface()
Для надёжности - объявляем виртуальный деструктор. В C++ возможно множественное наследование :
class Square : public Figure, public IDrawable ...

3. void TestMultipleInheritance()
Лучше избегать множественно наследования. Эта фраза не относится к интерфейсам.
Рассмотреть пролему ромба(diamond problem), решение при помощи виртуального наследования

4. void TestCppHpp()
Разделение класса на файлы реализации и интерфейса.
File->New->Class
Для простоты поместить cpp и h файлы в корень проекта(по дефаулту визард ставит в src.include папки), там где main.cpp. Часто для c++ используют hpp файлы, так сразу видно где C, а где C++ заголовочный фаил.
Директива #pragma once, да и вобще что такое #pragma

6. Рассмотреть пример с Аллегро :
- новое класс AllegroBase
- абстрактный метод Draw
- наверника будет проблема с
const int SCREEN_W = 640; // в cpp файлах
const int SCREEN_H = 480; // в cpp файлах
Можно их сохранить в самой фигуре или передовать методам которым она нужна. На след. занятии поговорим о других способах.
В примере опять нету виртуального деструктора. Может на это раз кто-то заметит

Практика:
0) переделать проект с прошлого занятия:
- AllegroBase
- абстрактные методы
- разбить на cpp/h файлы

Продолжить с предыдущего:
0) Показать как создать класс через визард, для простоты пусть заголовочный и реализации будет в корне проекта(тогда не надо прописывать компилятору путь к загаловочным файлам)
1) Добавить Figure -> MoveableFigure, SizeableFigure.
MoveableFigure->MoveableSquare, MoveableCircle
SizeableFigure ->(меняет своё размер, то есть если в MoveableFigure меняются координаты, то в SizeableFigure меняется коэфициент размера(для круга радиус, для квадрата сторона) SizeableCircle, SizeableSquare
Например, круг у которого меняется радису(уменьшается до 20% и затем увеличивается). Для начала можно о столкновениях не думать, но потом надо и это реализовать
План:
1.1) подключить AllegroBase класс
1.2) сделать иерархию Moveable*
1.3) разбить всё кроме AllegroApp на файлы. Тут есть проблема с константами типа SCREEN_W - пусть сделают отдельный фаил config.h засунут всё это туда - и затем в самом начале main сделать include этого файла - это не правильно, но на этом занятие пускай так. На следующем расскажу про enum
2) Сделать FigureList класс потомком от Figure и заставить перемещатся так же как и остальные фигуры. Для этого нужно сделать чтобы FigureList имел свои границы(для простоты пускай это квадрат) и передовал их своим объектам и фигуры бы рисовались относительно границ FigureList.
3) Добавить прямоугольник который переползает(подтягивает свой край к другому краю)
4) Подумать о взаимодействии фигур между собой. Например сталкивание между собой, можно сделать CollisionManager класс, который будет проверять столкнование и если да, то инвертить скорость, пусть базавая фигура реализует ещё и ICollidable , GetCollisionRange - и если фигуры в радиусе, то считать что столкновение. Можно и более сложное столкновение

```
174 void ChangeIntPointer( int *x )
175 {
176     *x = 100;
177 }
178
179 void ChangeIntRef( int &x )
180 {
181     x = 200;
182 }
183
184 void Caller3( Parent &p )
185 {
186     p.M1();
187     p.M2();
188 }
189
190 void TestReferences()
191 {
192     int x;
193     ChangeIntPointer( &x );
194     cout << x << endl;
195     ChangeIntRef( x );
196     cout << x << endl;
197
198     int *pX = &x;
199     *pX = 300;
200     cout << x << " " << *pX << endl;
202     int &refX = x;
203     refX = 400;
204     cout << x << " " << refX << endl;
205
206    Parent p;
207    Child c;
208    Caller3( p );
209    Caller3( c );
210    Parent *p2 = new Child();
211    Caller3( *p2 );
212    delete p2;
213    p2 = 0;
214    //Caller3( *p2 ); // runtime error when call M2(because
virtual)
```

```
215    int *pA = 0;
216    int &rA = *pA;
217    // cout << rA << endl; // runtime error
218 }
```

```
001 class Figure
002 {
003 public:
004     virtual ~Figure(){};
005     virtual void Draw() = 0;
006 };
007
008 class Square : public Figure
009 {
010 public:
011     virtual void Draw()
012     {
013         cout << "Square draw" << endl;
014     }
015 };
016
017 void Draw( Figure &f )
018 {
019     f.Draw();
020 }
021
022 void TestAbstract()
023 {
024     // Can't, Figure is abstract class
025     //Figure f;
026     Square sq;
027     sq.Draw();
028     Figure &f = sq;
029     f.Draw();
030     Draw( sq );
031 }
032
033 class IPrintable
034 {
035 public:
036     virtual ~IPrintable(){};
037     virtual void Print() = 0;
038 };
039
```

```
040 class Book : public IPrintable
041 {
042 public:
043     virtual void Print()
044     {
045         cout << "Printing book content ..." << endl;
046     }
047 };
048 class Circle : public Figure, public IPrintable
049 {
050 public:
051     virtual void Draw()
052     {
053         cout << "Square draw" << endl;
054     }
055
056     virtual void Print()
057     {
058         cout << "Printing circle content ..." << endl;
059     }
060 };
061
062 void PrintSomething( IPrintable &p )
063 {
064     p.Print();
065 }
066
067 void TestInterface()
068 {
069     Book b;
070     b.Print();
071     IPrintable &p = b;
072     p.Print();
073     PrintSomething( b );
074     Circle c;
075     PrintSomething( c );
076 }
```

```
078 class Person
079 {
080 public:
081     void GetName()
082     {
083         cout << "I have name" << endl;
084     }
085 };
086 class Worker
087 {
088 public:
089     void GetSalary()
090     {
091         cout << "I have salary" << endl;
092     }
093 };
094 class Developer : public Person, public Worker
095 {
096 };
097
098 class A
099 {
100 public:
101    int foo()
102    {
103        return 1;
104    }
105 };
106 class B: public virtual A
107 {
108 };
109 class C : public virtual A
110 {
111 };
112 class D : public B, public C
113 {
114 };
115
116 void TestMultipleInheritance()
117 {
118     Developer dev;
119     dev.GetName();
120     dev.GetSalary();
121
122     D d;
123     d.foo();
124 }
```

```
//-- Parent.h
01 #ifndef PARENT_H
02 #define PARENT_H
03
04 class Parent
05 {
06 public:
07     Parent();
08     virtual ~Parent();
09
10     void M1();
11     virtual void M2();
12 };
13
14 #endif // PARENT_H
```

```
//-- Parent.cpp
01 #include "Parent.h"
02 #include <iostream>
04 using namespace std;
06 Parent::Parent()
07 {
08 }
10 Parent::~Parent()
11 {
12 }
14 void Parent::M1()
15 {
16     cout << "Parent::M1" << endl;
17 }
19 void Parent::M2()
20 {
21     cout << "Parent::M2" << endl;
22 }
```

```
//-- Child.h
1 #pragma once
2
3 #include "Parent.h"
4 class Child : public Parent
5 {
6 public:
7     Child();
8     void M1();
9     virtual void M2();
10};
//-- Child.cpp
01 #include "Child.h"
02 #include <iostream>
04 using namespace std;
05 Child::Child() :
06    Parent()
07 {
08 }
09 void Child::M1()
10 {
11     cout << "Child::M1" << endl;
12 }
13 void Child::M2()
14 {
15     cout << "Child::M2" << endl;
16 }

//-------------------------------------------------------------
01 #include "Child.h"
02 void TestCpp()
03 {
04     Parent p;
05     p.M1();
06     p.M2();
07
08     Parent *p2 = new Child();
09     p2->M1();
10     p2->M2();
11     delete p2;
12 }
```

```
// -- AllegroBase.h
01 #pragma once
02
03 #include <allegro5/allegro.h>
04 #include <allegro5/allegro_primitives.h>
05
06 class AllegroBase
07 {
08     public:
09         AllegroBase();
10         virtual ~AllegroBase();
11
12         bool Init( int screenWidth, int screenHeight, int
fps );
13         void Destroy();
14         void Run();
15         void Exit();
16
17         virtual void Fps() = 0;
18         virtual void Draw() = 0;
19
20     protected:
21         ALLEGRO_DISPLAY *alDisplay_;
22         ALLEGRO_EVENT_QUEUE *alEventQueue_;
23         ALLEGRO_TIMER *alTimer_;
24
25     private:
26         bool exit_;
27
28 };

// -- main.cpp, unimportant parts are removed
01 #include "AllegroBase.hpp"
02 class Figure
03 {
04 public:
05     void Reset() { ... }
06     virtual void Draw() = 0;
07     virtual void Move() { ... };
08 };
09 typedef Figure * PFigure;
```

```
10
11 class Square : public Figure
12 {
13 public:
14     Square( double a ) { ... }
15     virtual void Draw() { ... }
16 };
17
18 class Circle : public Figure
19 {
20 public:
21     Circle( double r ) { ... }
22     virtual void Draw(){ ... }
23 };
24
25 class ScreenSaver
26 {
27 private:
28     PFigure figures[MAX];
29     int size_;
30 public:
31     ScreenSaver() { ... }
32     void Draw()
33     {
34         al_clear_to_color( al_map_rgb( 0, 0, 0 ) );
35         for( int i = 0; i < size_; ++i )
36         {
37             figures[i]->Draw();
38         }
39     }
40
41     void Next()
42     {
43         for( int i = 0; i < size_; ++i )
44         {
45             figures[i]->Move();
46         }
47     }
48
49     void Add( Figure *f ) { ... }
50 };
```

```
52 class AllegroApp : public AllegroBase
53 {
54 private:
55     ScreenSaver ss;
57 public:
58     AllegroApp() :
          AllegroBase(),
59        ss()
60     {
61         for( int i = 0; i < 100; ++i )
62         {
63             if ( ( i % 2 ) == 0 )
64             {
65                 ss.Add( new Circle( 10.0 + rand() % 30 ) );
66             }
67             else
68             {
69                 ss.Add( new Square( 10.0 + rand() % 30 ) );
70             }
71         }
72     }
74     virtual void Fps()
75     {
76         ss.Next();
77     }
79     virtual void Draw()
80     {
81         ss.Draw();
82     }
83 };
85 int main(int argc, char **argv)
86 {
87     srand( time(0) );
89     AllegroApp app;
90     if ( !app.Init( SCREEN_W, SCREEN_H, FPS ) )
91     {
92         return 1;
93     }
94     app.Run();
97     return 0;
98 }
```

5. Статические методы, Singleton, Factory

В C++ метод можно сделать статическим(классовым). Эти методы не имеют доступа к данным объекта и для их использования не нужно создавать экземпляры.

1. void TestStatic()
Счётчик объектов. синтаксис статик данных: декларация\размещение; синтаксис статик методов(внутри объявления класса, вне объявления)
Статические методы наследуются, но они не полиморфны(не могут быть виртуальными).
Конструктор - это почти статический методы, не полиморфный, вызов без экземпляра.

2. void TestStaticClass ()
Выделить набор ф-ций в отдельный класс чтобы избежать коллизий имён. Например в джаве есть класс System. В C++ можно сделать подобное. Основное назначение чисто статических классов в группировке логически схожих методов, констант, полей и свойств.

3. void TestSingleton()
// простая реализация, не совсем правильная(не закрытый копи конструктор, оператор присваивания, не тредсайф), но о более правильной потом. тут сама идея видна.
Синглетон - всегда один экземлпяр, доступ глобальный. Возможно передать в качестве параметра ф-ции(статик классы нельзя), поддерживает интерфейсы. Легче тестировать чем статик классы со статик данными. Чтобы сократить код для доступа к синглтону можно сделать хелпер :

```
Preferences & MainPref ()
{
    return Preferences::Instance()
};
```

4. void TestFactory()
Фабрика(Factory method pattern) - позволяет создовать объекты(или даже группы) с определёнными настройками(абстрагируют процесс инстанцирования).
В общем случае для фабрики необязательно выделять целый класс, иногда може просто сделать фабричный метод.
Есть ещё абстрактная фабрика. Например есть фабрика для создания графических элементов. Всё хорошо когда поддерживатеся одна платформа. Если надо добавить другую(например есть для десктопов, нужно добавить для мобильных устройств), то таже кнопка на десктопе и на мобильном у-ве будет разная(одна принимает щелчки мыши(WinButton), другая реагирует на тачь(TouchButton)) и соотвественно нужны две фабрики с общим интерфейсом CreateButton())
http://en.wikipedia.org/wiki/Abstract_factory_pattern

5. void TestFriends()
Дружественные классы или ф-ции имеет доступ к защищённой и приватной секции.
Большое количество дружественных классов - признак плохо проектирования, так как дружба нарушает инкапсуляцию. Но если класс знает что он делает, то дружба возможна. Например та же фабрика фигур может быть другом квадарата, сделать конструктор квадрата приватным, и тогда есть возможность разрешить только Красные или Синие квадраты. Или например есть класс Фигура, и есть класс МенеджерФигур, он может быть дружественным у Фигуры для прямых взаимодействий(оптимизации и т.п.)

6. Пример с Аллегро.
ScreenSaver стал синглтоном, появилась фабрика фигур, перехват клавиш через Аллегро.

Есть упоминание о enum'e - о нём очень кратко, более подробно на другом занятии

Практика:
У многих фигуры уже отталкиваються друг от друга
0) Сделать какю-то простую игрушку, например:
Есть круг, которым управлеяет человек, надо уворачиватся от фигур. При столкновение проигрышь. Посчитать сколько секунд выжил(ну или для простоы сколько тиков выжил)

Что нужно улучшить. Сделать фабрику для фигур, сделать синглетон для игрока(внутри может быть фигура для отображения, жизни, специал абилити и т.д.). Если нужна какая-то геометрия, выделить в отдельный класс со статик методами bool Physics::IsCollided(Figure *f1, Figure *f2);

1) добавить сложность, каждые 500 тиков скорость фигур увеличивается на 5%. Каждые 1000 тиков появляется новая фигура.
2) добавить жизни
3) добавить невидимость при нажатие на пробел. Невидимость должна быть ограничена, например, 5 раз на 400 тиков. Невидимость должна визуально отображаться, например в начале видны только контуры, а к концу весь(а-ля градиент с 0 до 255) или заюзать альфа канал

```
001 class Book
002 {
003 public:
004     Book()
005     {
006         ++count_;
007     }
008     ~Book()
009     {
010         --count_;
011     }
012     static int GetCount();
013     //static int GetCount()
014     //{
015     //   return count_;
016     //}
017 private:
018     static int count_;
019
020 };
021 int Book::count_ = 0;
022 int Book::GetCount()
023 {
024     return count_;
025 }
026
027 void TestStatic()
028 {
029     cout << Book::GetCount() << endl;
030     Book b;
031     cout << Book::GetCount() << endl;
032     {
033         Book bb[10];
034         cout << Book::GetCount() << endl;
035     }
036     cout << Book::GetCount() << endl;
037 }
038
039 class Math
040 {
041 public:
042     static double Sqr( double x )
043     {
044         return x * x;
045     }
046     static double C( double a, double b )
047     {
048         return sqrt( Sqr( a ) + Sqr( b ) );
049     }
050 };
051
052 void TestStaticClass()
053 {
054     cout << Math::C( 3, 4 ) << endl;
056 }
057
058 class Preferences
059 {
060 public:
061     static Preferences &Instance()
062     {
063         // Guaranteed to be lazy initialized
064         static Preferences instance;
065         return instance;
066     }
067
068     int GetMaxUsers()
069     {
070         return maxUsers_;
071     }
072
073 private:
074     Preferences() :
075         maxUsers_( 10 )
076     {};
077     int maxUsers_;
078 };
079
080 void TestSingleton()
081 {
082     cout << Preferences::Instance().GetMaxUsers() << endl;
083 }
```

```
086 class Figure
087 {
088 public:
089     virtual ~Figure(){};
090     virtual void Draw() = 0;
091 };
092
093 class Square : public Figure
094 {
095 public:
096     Square( double side, double color ) :
097         side_( side ),
098         color_( color )
099     {
100     }
101     virtual void Draw()
102     {
103         cout << "Square draw" << endl;
104     }
105 private:
106     double side_;
107     double color_;
108 };
109
110 class Circle : public Figure
111 {
112 public:
113     Circle( double r ) :
114         r_( r )
115     {
116     }
117     virtual void Draw()
118     {
119         cout << "Circle draw" << endl;
120     }
121 private:
122     double r_;
123 };
124
125 void Draw( Figure &f )
126 {
127     f.Draw();
128 }
129
130 class FigureFactory
131 {
132 public:
133     static Figure *CreateCircle( double r )
134     {
135         return new Circle( r );
136     }
137     static Figure *CreateEarth()
138     {
139         return new Circle( 63e9  );
140     }
141     static Figure *CreateRedSquare( double side )
142     {
143         return new Square( side, 0xFF0000 );
144     }
145     static Figure *CreateSmallGreenSquare()
146     {
147         return new Square( 1e-5, 0x00FF00 );
148     }
149 };
150
151 void TestFactory()
152 {
153     Figure *f[5];
154     f[0] = new Circle( 10 );
155     f[1] = FigureFactory::CreateCircle( 10 );
156     f[2] = FigureFactory::CreateEarth();
157     f[3] = FigureFactory::CreateRedSquare( 10 );
158     f[4] = FigureFactory::CreateSmallGreenSquare();
159     for( int i = 0; i < 5; ++i )
160     {
161         f[i]->Draw();
162         delete f[i];
163         f[i] = 0;
164     }
165 }
```

```
167 class A
168 {
169     friend class B;
170     friend void Foo( A &a );
171 private:
172     int x_;
173     void ChangeX()
174     {
175         x_ = 10;
176     }
177 };
178 class B
179 {
180 public:
181     static void ChangeA( A &a )
182     {
183         a.x_ = 12;
184         cout << a.x_ << endl;
185     }
186 };
187
188 void Foo( A &a )
189 {
190     a.ChangeX();
191     cout << a.x_ << endl;
192 }
193
194 void TestFriends()
195 {
196     A a;
197     B::ChangeA( a );
198     Foo( a );
199 }
```

```
001 class AllegroBase
002 {
003 public:
004     // See help for ALLEGRO_KEYBOARD_STATE
005     bool IsPressed( int keycode );
006     virtual void OnKeyDown
                ( const ALLEGRO_KEYBOARD_EVENT &keyboard ) {};
007     virtual void OnKeyUp
                ( const ALLEGRO_KEYBOARD_EVENT &keyboard ) {};
008 };
009
010 class Figure
011 {
012 protected:
013     double x_;
014     double y_;
015     double dx_;
016     double dy_;
018 public:
019     ...
020     virtual void Move()
021     {
022         x_ += dx_;
023         y_ += dy_;
024         if ( ( x_ < 1.0 ) ||
025              ( x_ > SCREEN_W ) ||
026              ( y_ < 1.0 ) ||
027              ( y_ > SCREEN_H ) )
028         {
029             Reset();
030         }
031     };
032 };
035 class ScreenSaver
036 {
037 private:
038     PFigure figures[MAX];
039     int size_;
040 public:
041     static ScreenSaver &Instance()
042     {
043         static ScreenSaver instance;
044         return instance;
045     }
046     void Draw() { ... }
047     void Next() { ... }
048     void Add( Figure *f ) { ... }
049     void Reset()
050     {
051         for( int i = 0; i < size_; ++i )
052         {
053             figures[i]->Reset();
054         }
055     }
056 private:
057     ScreenSaver() :
058         size_( 0 ) { ... }
059     ~ScreenSaver() { ... }
060 };
061
062 class ControlledSquare : public Square
063 {
064 public:
065     ControlledSquare( double side ) :
066         Square( side )
067     {
068     }
069     void MoveBy( double dx, double dy )
070     {
071         dx_ = dx;
072         dy_ = dy;
073         Move();
074     }
075 };
077 class FigureFactory
078 {
079 public:
080     enum Type
081     {
082         RandomCircle,
083         RandomSquare
084     };
```

```
085
086     static Figure * Create( Type type )
087     {
088         switch( type )
089         {
090             case RandomCircle:
091                 return new Circle( 10.0 + rand() % 30 );
092             case RandomSquare:
093                 return new Square( 10.0 + rand() % 30 );
094         }
095     }
096 };
097
098 class AllegroApp : public AllegroBase
099 {
100 private:
101     ControlledSquare humanSquare_;
102
103 public:
104     AllegroApp() :
105         AllegroBase(),
106         humanSquare_( 30 )
107     {
108         for( int i = 0; i < 10; ++i )
109         {
110             if ( ( i % 2 ) == 0 )
111             {
112 ScreenSaver::Instance().Add(
113     FigureFactory::Create( FigureFactory::RandomSquare ) );
114             }
115             else
116             {
117 ScreenSaver::Instance().Add(
118     FigureFactory::Create( FigureFactory::RandomCircle ) );
119             }
120         }
121     }
122
123     virtual void Fps()
```

```
124     {
125         ScreenSaver::Instance().Next();
127         double dx = 0, dy = 0;
128         if ( IsPressed( ALLEGRO_KEY_UP ) )
129         {
130             dy = -1;
131         }
132         if ( IsPressed( ALLEGRO_KEY_DOWN ) )
133         {
134             dy = +1;
135         }
136         if ( IsPressed( ALLEGRO_KEY_LEFT ) )
137         {
138             dx = -1;
139         }
140         if ( IsPressed( ALLEGRO_KEY_RIGHT ) )
141         {
142             dx = +1;
143         }
144         if ( IsPressed( ALLEGRO_KEY_LSHIFT ) )
145         {
146             dy *= 10;
147             dx *= 10;
148         }
149         humanSquare_.MoveBy( dx, dy );
150     }
152     virtual void Draw()
153     {
154         ScreenSaver::Instance().Draw();
155         humanSquare_.Draw();
156     }
158     virtual void OnKeyDown
                ( const ALLEGRO_KEYBOARD_EVENT &keyboard )
159     {
160         if ( keyboard.keycode == ALLEGRO_KEY_SPACE )
161         {
162             ScreenSaver::Instance().Reset();
163         }
164     }
165 };
```

6. Namespace, std::string

namespace - пространство имён — некоторое множество каким-либо образом взаимосвязанных имён или терминов. В программирование идентификаторы внутри пространства имён не должны повторятся. Если название города взять как имя пространства, то улицы - это индентификаторы, в разных города могут быть улицы с одинаковым названием, но в пределах одного города - нет.
1 . void TestNamespaces
В пространство можно поместить всё что угодно(не только ф-ции)
Они могут объединятся(пример с cpp и h файлами)
Они могут быть вложенными, тогда доступ space1::space2::identifactor
глобальное пространство.
У имён могут быть псевдонимы, типа
namespace ATT = American_Telephone_and_Telegraph;
ATT::Foo()
using namespace не рекомендуется использовать в загаловочных фаилах, так как это может привести к измененю поведения кода который включает этот фаил. То есть namespace так же распространиться и на сам фаил имплементации, то есть в заголовочных фаилах нужно писать std::string

2. void TestString();
std::string - это как бы массив из char в виде объекта. Строки хранят свой буффер в куче. Предоставлют автоматическое расширение\уменьшение объёма памяти для хранения. std::string предоставляют совместимость с C-string, многие операторы и методы принимают C-string.
Оператор []
Поиск в строке, индикатор string::npos.
В c++ есть wstring для работы например с UTF-8, но прямых средств нету. wstring служит для хранения w_char(по 2 или 4 байта на символ(зависит от ОС)), не привязан к кодировки, поэтому связь с UTF нужно реализовывать вручную

3. void TestMan()
Для форматного вывода можно использовать
cout.precision(int)
cout.width(int)
cout.fill(char)
Но это не удобно, так их нельяз в одну строку написать.
Манипуляторы делают это более удобным. Некторые манипуляторы меняют только до следующего вывода, некоторые имеют постоянный эффект.

4. void TestStrNum()
Потоковое преобразование. Можно преобразовывать при помощи sprintf, sscanf, но для этого нужны с-строки. Поэтому не удобно. А вот потоки позволяют преобразовывать точно так же как при выводе на экран. Можно использовать манипулятор

5. void TestConst()
\\ на это времение скорея всего не хватит, поэтому хотя бы поговорить по поводу передачи строк : void Foo(string s) - будет полная копия, не очень хорошо. Нужно использовать const или просто ссылки.

const ссылки, для setter и результат getter

copy constructor, const экземляры класса и const методы. У const аргументов можно вызывать const методы. А const методы не могут менять объект(пока про mutable не надо).

Практика:
1) что-то со строками. Например серилиазовать объект в строку и вывести на экране(в файл, но лучше пока без файлов)
Square:x=100,y=120,side=5; Circle:x=10,y=120,side=20;
2) десереализовать : получить из строки объект.
\\ для парсинга предполагаем что нет пробелов, можно использовать в начале find(";"), делаем подстроку с объектом, создаём его(find(":")), и остаток прокидываем в сам объект.
там find("=") и find(",")
у find есть аргумент для смещения
Например можно так, если пользователь нажимает клавишу P - то все объекты печатаются на консоли.
Если пользователь нажимает клавишу I - то он может ввести что-то из консоли и тем самым создать объект
Пускай юзают интерфейс ISerializable : ToString/FromString
3) пускай дальше игрушку развивают(как описано на предыдущем занятии)

```
http://www.cplusplus.com/reference/string/
001 void Foo()
002 {
003     cout << "global" << endl;
004 }
005
006 namespace space1
007 {
008
009 void Foo()
010 {
011     cout << "space1" << endl;
012 }
013
014 }
015
016 namespace space2
017 {
018
019 void Foo()
020 {
021     cout << "space2" << endl;
022 }
023
024 }
025
026 // in some.h
027 namespace space3
028 {
029 struct SomeClass
030 {
031     SomeClass();
032 };
033 }
034 // in some.cpp
035 namespace space3
036 {
037 SomeClass::SomeClass()
038 {
039 }
040 }
```

```
041 using space3::SomeClass;
043 namespace space4
044 {
045 double sqrt(double x)
046 {
047     return x;
048 }
049 double sin(double x)
050 {
051     return x;
052 }
053 }
054 using namespace space4;
055
056 void TestNamespaces()
057 {
058     Foo();
059     space1::Foo();
060     space2::Foo();
061     space3::SomeClass c;
062     SomeClass c2;
063     cout << space4::sqrt( 4.0 ) << endl;
064     cout << ::sqrt( 4.0 ) << endl;
065 }
066
067 void TestString()
068 {
069     string s = "abc";
070     cout << s << endl;
071     s += "def";
072     printf( "as c-string %s\n", s.c_str());
073     const char *c = "12345";
074     if ( s != c )
075     {
076         cout << s << " not equal " << c << endl;
077     }
079     for( int i = 0; i < s.length(); ++i )
080     {
081         cout << s[i];
082         s[i] = toupper(s[i]);
083     }
```

```
084     cout << endl << s << endl;
085
086     string s1 = "123456";
087     cout << "found at 56 at " <<  s1.find( "56" ) <<
endl;
088     size_t pos = s1.find( "abc" );
089     if ( pos == string::npos )
090     {
091         cout << "abc not found " << endl;
092     }
093     getline( cin, s );
094     cout << "entered = " << s << endl;
095     cin >> s; // till space
096     cout << "entered = " << s << endl;
097 }
098 /* Most usefull :
099 constructors
100 assign - Assign content to string
101 append - Append to string
102 insert - Insert into string
103 erase - Erase characters from string
104 replace - Replace portion of string
105 substr - Generate substring
106 copy - Copy sequence of characters from string to C-
string
107 compare - Compare strings
108 find - Find content in string
109 rfind - Find last occurrence of content in string
110
111 For chars <ctype>:
112 is*( like isdigit, isalpha, isspace ...
113 toupper, tolower
114 */
115
116 #include <iomanip>
117 void TestMan()
118 {
119     cout << setfill('_') << setw(6) << "six" << "none"
<< endl;
120
121     double x = 123.45678;
122     // default
123     cout << x << endl;
124     // Maximum number of digits to display
125     cout << setprecision( 4 ) << x << endl;
126     // Two digits after .
127     cout << fixed << setprecision( 2 ) << x << endl;
128     // Output in oct/dec/hex for integers
129     cout << setbase( 16 ) << 32 << endl;
130
131     // Note: setprecision and setbase have constant
effect
132     cout << 567.8 << " " << 16 << endl;
133
134     int saveP = cout.precision();
135     cout << setprecision(4) << 123.4567 << endl;
136     cout.precision( saveP );
137     cout << 123.4567 << endl;
138 }
139
140 #include <sstream>
141 void TestStrNum()
142 {
143     string s = "123.45";
144     istringstream is( s );
145     double x;
146     is >> x;
147     cout << x << endl;
148
149     x = 123.45;
150     ostringstream os;
151     os << fixed << setprecision(1) << x;
152     s = os.str();
153     cout << s << endl;
154 }
155
```

```
156 class Data
157 {
158 private:
159     int x_;
160     string s_;
161 public:
162     Data( int x ) :
163         x_( x ),
164         s_()
165     {
166
167     }
168   Data( const Data &other ) :
169         x_( other.x_ ),
170         s_( other.s_ )
171   {
172   }
173   static const int MAX_SIZE;
174   int GetX() const
175   {
176       return x_;
177   }
178   void SetX( int x )
179   {
180       x_ = x;
181   }
182   const string &GetStr() const
183   {
184       return s_;
185   }
186   void SetStr( const string &s )
187   {
188       s_ = s;
189   }
190 };
191 const int Data::MAX_SIZE = 10;
192
```

```
193 void CallOnConst( const Data &d )
194 {
195     cout << d.GetX() << d.GetStr() << endl;
196     // d.SetX( 11 ); // error: passing 'const ...
197 }
198
199 void TestConst()
200 {
201     Data x( 0x10 );
202     Data y = x; // Copy constructor call
203     y.SetStr( "asd" ); // will create string from "asd"
204                        // and pass this to SetStr
205     CallOnConst( y );
206 }
```

7. Const для классов, контейнеры, итераторы

1. void TestConst()
const ссылки, для setter и результат getter
copy constructor, const экземляры класса и const методы. У const аргументов можно вызывать const методы. А const методы не могут менять объект(пока про mutable не надо).

Стандартные контейнеры. Можно их самих запрограммировать, но в C++ пошли навстречу. Спросить какие бы структуры данных хотели бы видеть в библиотеке C++ и определить что есть, а что всё же придётся программировать самому.
vector - Dynamic array
list - Linked list
set - Set
map - Associative arrays
stack - LIFO stack
queue - FIFO queue
bitset - Bitset
deque - Double ended queue
priority_queue - Priority queue
multiset - Multiple-key set
multimap - Multiple-key map

2. void TestVector
Стандартные массивы из C использовать не очень удобно, в C++ есть вектор, который свои данные хранит в куче. Вектор автоматически управляет своей памятью.
Вектор хранит свою память как неразрывный кусок памяти(так же как и массивы C). Это значит что при добавление\удаление возможно реаллоцирование элементов. Доступ к элементам за константное время. У вектора есть зарезервированное место для роста(capacity) - то есть не каждое добавление элемента видёт к реаллокации. Это означает что в сравнение с C, вектор занимает не много больше, например, для хранении 1000 элемента, будет выделено место 1024(это управляемый процесс(управление не очень удобно, но возможно), так что программист может управлять размером резервации, но обычно это не проблема)
Можно присвоивать один массив в другой(самое главное чтобы тип элемента был одинаковый)

3. void TestList
Списки - контейнеры с последовательным хранение данных. Вставка\удаление элемента происходит за константное время, поиск за O(n) . Реализован как двунаправленный список(есть forward_list(c++11) - одноправленный, в результате немного меньшего размера, немного быстрея, но пробежаться можно только в одну сторону)
Список использует когда нужно часто менять элементы(удалять, вставлять, менять). Например для сортировки. Нету прямого доступа, поэтому доступ к I-томы элементу за I-операций перехода.
В кратце о итераторах - это вспомогательный объект который указывает на другой объект(данные). Идея в том чтобы итератор мог перейти на следующий, на предыдущий и дать доступ к текущему(указываемому) элементу. Итератор похож на указатель завраплленный в объект(то есть итератор очень лёгкий объект)

Спросить что выведет код

2-ой for: 4 3 2 1 0
3 for: 8 6 4 2 0
4 for: Обяснить почему так нелья. (begin() - итератор на первый элемент, end() - итератор на конец(но это не последний элемент, это как бы за последним). примерно как и for(int i < 0; i < 5; ++i), выведет от 0 до 4 (а не до 5)
5 for: если идти с конца то есть reverse_iterator, const_reverse_iterator. Смысол как и у обычных операторов, только для реверсионного перейти на следующий элемент - это в памяти перейти на предыдущий. и соотвественно rbegin() - итератор на последний элемент, rend() - итератор на элемент перед первым
6 for: Итераторы можно двигать. Пускай догадаются что выведет.
7 for: Конструкторы контейнеров могут принимать итераторы. То есть контейнеры могут перебрасывать элементы при помощи итераторов. Вектор создаётся на основе списка. И Тоже пускай догадаются что выведет.
8 и 9 for: простой пример, например вставка.Но поговорить по поводу "Iterator validity". Надо понимать что мы делаем. Для списка правила одни, для вектора правила другие. Думайте об итераторе как о указателе, если происходит реаллокация - то все итераторы становятся не валидными, это головная боль программиста, библиотека не защитит от доступа по не валидному итератору(эффект похож на указатель в случайную память)
Что выведут :
100 100 200 100 100
100 100 200 100 100 300 300
10, 11 for : та же валидность, можно и так. Что выведет:
13 100 13 100 13 200 13 100 13 100 13 300 13 300
Спросить можно ли такое для вектора.

4. void TestMap \\ скорея всего на это не будет времени, поэтому к пункту 5
Это ассоциативный массив, ключ - значение. В качестве значения может выступать любой объект(с дефолтным конструктором). В качестве ассоциации может выступать любой объект, который можно сравнивать(например operator<)
Внутренне это бинарное дерево поиска(обычно, хотя и не обязательно). Особенность в том что есть опертатор прямого доступа по ключу(сложность логарифмическая от размера)
Элементы - это пара(ключь, значение).
Вставка очень простая. Оператор [] создаст(дефолтный конструктор) значени по ключу если его нету. И затем присвоит новое значение
Внимание поиск, через == делать не надо. Объяснить почему(первое обращение сосдаст значение по ключу, это может привести к ошибкам). Объяснить как надо.

5. void TestPolymorphism
Полиморфизм и контейнеры - отсутсвует, надо вручную например вот так. Обычно всё это врапится в какой-то класс а-ля менеджер объектов.
+ если через итератор, то
(*it)→Draw();
не заюбудьте что если использовать std классы в загловочных фаилах, то нужно явно указать namespace
std::vector, std::string

Практика:
Переписать на list< Figure *> и итераторы. Использовате typedef для контейнера и итераторов на него.
(*it)->Draw();
или

```
Figure *f = *it;
f->Draw();
f->...ы
```

И продолжить какие-нибудь фишки добавлять(см. предыдущие лекции)

```
001 class Data
002 {
003 private:
004     int x_;
005     string s_;
006 public:
007     Data( int x ) :
008         x_( x ),
009         s_()
010     {
012     }
013   Data( const Data &other ) :
014         x_( other.x_ ),
015         s_( other.s_ )
016   {
017   }
018   static const int MAX_SIZE;
019   int GetX() const
020   {
021       return x_;
022   }
023   void SetX( int x )
024   {
025       x_ = x;
026   }
027   const string &GetStr() const
028   {
029       return s_;
030   }
031   void SetStr( const string &s )
032   {
033       s_ = s;
034   }
035 };
036 const int Data::MAX_SIZE = 10;
037
038 void CallOnConst( const Data &d )
039 {
040     cout << d.GetX() << d.GetStr() << endl;
041     // d.SetX( 11 ); // error: passing 'const ...
042 }
```

```
044 void TestConst()
045 {
046     Data x( 0x10 );
047     Data y = x; // Copy constructor call
048     y.SetStr( "asd" ); // will create string from "asd"
049                        // and pass this to SetStr
050     CallOnConst( y );
051 }
053 #include <vector>
054 void TestVector()
055 {
056     int aInts[1000];
057     vector< int > vInts(1000);
058     cout << sizeof( aInts ) << endl;
059     cout << sizeof( vInts ) << endl; // 12 bytes
060     for( int i = 0; i < 10; ++i )
061     {
062         vInts[i] = i * i;
063         cout << vInts[i] << endl;
064     }
065     vector <int> v; // 0 size
066     cout << v.size() << " " << v.capacity() << endl;
067     for( int i = 0; i < 10; ++i )
068     {
069         v.push_back( i );
070         cout << v.size() << " " << v.capacity() <<endl;
071     }
072     cout << v.front() << " " << v.back() << endl;
073     /*  0 0
074         1 1
075         2 2
076         3 4
077         4 4
078         5 8
079         6 8
080         7 8
081         8 8
082         9 16
083         10 16
084         0 9 */
```

```
085     vInts = v;
086     cout << vInts.back() << endl; // 9
087 }
088 /*
089 size Return size
090 empty Test whether vector is empty
091 resize Change size
092 reserve Request a change in capacity
093 shrink_to_fit Shrink to fit(Cx11 only)
094 front Access first element
095 back Access last element
096 push_back Add element at the end
097 pop_back Delete last element
098 insert Insert elements
099 erase Erase elements
100 swap Swap content(sizes may differ)
101 clear Clear content */
102
104 #include <list>
105 #include <iterator>
106 void TestList()
107 {
108     list< int > ints;
109     for( int i = 0; i < 5; ++i )
110     {
111         ints.push_front( i );
112     }
113     // for 2
114     for( list<int>::const_iterator it = ints.begin();
115          it != ints.end(); ++it )
116     {
117         cout << *it << endl;
118     }
119     // for 3
120     for( list<int>::iterator it = ints.begin();
121          it != ints.end(); ++it )
122     {
123         *it =  *it * 2;
124         cout << *it << endl;
125     }
126     // for 4, bad iterators can be iterated back
127     // prints <uninitialized memory, e.g 5> 0 2 4 6
128     for( list<int>::const_iterator it = ints.end();
129          it != ints.begin(); --it )
130     {
131         cout << *it << endl;
132     }
133     // for 5, good, use reverse iterator
134     for( list<int>::const_reverse_iterator
           rit = ints.rbegin(); rit != ints.rend(); ++rit )
136     {
137         cout << *rit << endl;
138     }
139     // for 6
140     list< int >::iterator it3 = ints.begin();
141     advance( it3, 2 );
142     list< int > l2( it3, ints.end());
143     for( list<int>::const_iterator it = l2.begin();
144          it != l2.end(); ++it )
145     {
146         cout << *it << endl;
147     }
148     // for 7
149     vector< int > v2(ints.rbegin(), ints.rend());
150     for( vector< int >::const_iterator it = v2.begin();
151          it != v2.end(); ++it )
152     {
153         cout << *it << endl;
154     }
155     // for 8-9
156     vector<int> v3( 4, 100 );// 4 ints with 100 value
157     v3.insert( v3.begin() + 2, 200 );
158     for( vector< int >::const_iterator it = v3.begin();
159          it != v3.end(); ++it )
160     {
161         cout << *it << endl;
162     }
163     list<int> l3(2, 300);
164     l3.insert( l3.begin(), v3.begin(), v3.end());
165     for( list< int >::const_iterator it = l3.begin();
166          it != l3.end(); ++it )
167     {
```

```
168         cout << *it << endl;
169     }
170     // for 10-11
171     for( list< int >::iterator it = l3.begin();
172          it != l3.end(); ++it )
173     {
174         l3.insert( it, 13 );
175     }
176     for( list< int >::const_iterator it = l3.begin();
177          it != l3.end(); ++it )
178     {
179         cout << *it << endl;
180     }
181 }
182 /*
183 empty Test whether container is empty
184 size Return size
185 assign Assign new content to container
186 push_front Insert element at beginning
187 pop_front Delete first element
188 push_back Add element at the end
189 pop_back Delete last element
190 insert Insert elements
191 erase Erase elements
192 swap Swap content
193 resize Change size
194 clear Clear content
195 remove Remove elements with specific value
196 unique Remove duplicate values
197 sort Sort elements in container
198 merge Merge sorted lists
199 reverse Reverse the order of elements */
201 #include <map>
202 void TestMap()
203 {
204     typedef map<string, int> MyMap;
205     typedef map<string, int>::iterator MyMapIt;
206     typedef map<string, int>::const_iterator MyMapCIt;
207     typedef pair<string, int> MyMapEl;
208     MyMap m1;
210     m1["key1"] = 100;
211     m1["key2"] = 200;
212     for( MyMapIt it = m1.begin(); it != m1.end(); ++it)
213     {
214         MyMapEl el = *it;
215         cout << el.first << " = " << el.second << endl;
216     }
217     cout << m1["key1"] << endl;
218     // BAD compare
219     if ( m1["key3"] == 300 )
220     {
221         cout << "key3 yes 300" << endl;
222     }
223     if ( m1["key3"] == 0 )
224     {
225         cout << "key3 yes 0" << endl;
226     }
227     if (  m1.find( "key4" ) == m1.end() )
228     {
229         cout << "key 4 not found" << endl;
230     }
231     MyMapCIt itKey1 = m1.find( "key1" );
232     if ( itKey1 != m1.end() )
233     {
234         cout << "key 1 found" << endl;
235         if ( itKey1->second == 100 )
236         {
237             cout << "and equal 100" << endl;
238         }
239     }
240     m1["key2"] = 201;
241     MyMapIt itKey2 = m1.find( "key2" );
242     if ( itKey2 != m1.end() )
243     {
244         itKey2->second = 202;
245     }
246     cout << m1["key2"] << endl;
247 }
248 /*
249 empty Test whether container is empty
250 size Return container size
251 insert Insert elements
```

```
252 erase Erase elements
253 swap Swap content
254 clear Clear content
255 find Get iterator to element
256 count Count elements with a specific key */
259 class A
260 {
261 public:
262     virtual ~A() {};
263     virtual void PrintMe() = 0;
264 };
265 class B : public A
266 {
267     virtual void PrintMe()
268     {
269     cout <<"It is B at " << (unsigned int)this << endl;
270     }
271 };
272 class C : public A
273 {
274     virtual void PrintMe()
275     {
276     cout <<"It is C at " << (unsigned int)this << endl;
277     }
278 };
280 void TestPolymorphism()
281 {
282     typedef vector< A * > Objs;
283     Objs objs;
284     objs.push_back( new B() );
285     objs.push_back( new C() );
286     for( int i = 0; i < objs.size(); ++i )
287     {
288         objs[i]->PrintMe();
289     }
290     for( int i = 0; i < objs.size(); ++i )
291     {
292         delete objs[i];
293     }
294     objs.clear();
296 }
```

```
297 void TestPolymorphismIterators()
298 {
299    typedef list< A * > Objs;
300    typedef Objs::iterator ObjsIt;
301
302    Objs objs;
303    objs.push_back( new B() );
304    objs.push_back( new C() );
305    for(ObjsIt it = objs.begin();it != objs.end(); ++it)
306    {
307        A *a = *it;
308        a->PrintMe();
309
310        A &b = **it;
311        b.PrintMe();
312
313        delete a;
314    }
315 }
```

8. Map, set, algorithm

1. void TestMap
Это ассоциативный массив, ключ - значение. В качестве значения может выступать любой объект(с дефолтным конструктором). В качестве ассоциации может выступать любой объект, который можно сравнивать(operator<)
Внутренне это бинарное дерево поиска(обычно, хотя и не обязательно). Особенность в том что есть оператор прямого доступа по ключу(сложность логарифмическая от размера)
Элементы - это пара(ключ, значение).
Вставка очень простая. Оператор [] создаст(дефолтный конструктор) значени по ключу если его нету. И затем присвоит новое значение
Внимание поиск, через == делать не надо. Объяснить почему(первое обращение создаст значение по ключу, это может привести к ошибкам). Объяснить как надо.
Map как и другие контейнеры хранят копию(если копия не нужно, то нужно хранить указатель на объект + предоставит компаратор)

2. void TestSet()
Множество - контейнер для хранения уникальных элементов. Внутренне элементы отсортированны, это обеспечивает быстрый доступ. Нужен оператор сравнения(operator<). Обычно реализованы как дерево поиска.

3. void TestBitSet()
Множество битов - контейнер для хранения битов (например можно использовать для хранения включенных опций). Битсет можно инициализировать числом или строкой("0100110"), но это не особо устойчиво к изменениям логики. Например можно использовать енум.

4. void TestAlgorithm()
Алгоритмы - предоставляют более-менее стандартную ф-циональность над контейнером. Например у вектора нету метода поиска(так как невозможно его сделать эффективным), но если уж надо, чтобы не писать каждый раз цикл 0..(size-1) можно использовать алгоритм find, который по сути и делает этот цикл.
Алгоритмы позволяют искать по разным условиям, проверять, модифицировать, сортировать, вытаскивать части котейнеров, искать элементы(макс, мин), сливать котейнеры в один, составлять контейнеры на основе других.
1) find - для С-массива не очень хорош, нельзя проверить найден или нет(ну можно, но не удобно) Если find не нашёл то он вернёт последний элемент, для вектора это end()
(правая граница поиска) , для С-массива это та же правая граница, в данном случае это адресс 4-ого элемента, а не нулевой указатель.
2) find_if - Использует ф-цию + можно использовать объект компаратор(здесь как временный объект, но можно как и экземпляр класса). Преимущество объекта 1) можно объявить рядом(только надо включить опцию gcc -std=c++0x, иначе template argument for XXX uses local type) 2) можно иметь насколько угодно сложное состояние(например можно найти 3 последних равных элемента)
3) search - у строк есть такой же метод, но оптимизирован для строк. Это алгоритм позволяет найти под массив в массиве(или под контейнер в контейнере). Опять же чтобы не писать цикл в цикле.
4) for_each - не хотите писать begin() ... end() циклы, вот решение. + тут есть вложенный класс. Классы\структуры могут быть вложенными, доступ через :: из вне класса, внутри класса к вложенному доступ напрямую. + инты(простые типы) передают по значению, но с объектами - это плохо, так как объект может быть большим. Решает это константные

ссылки. Можно написать объект компаратор или статик метод(по сути это глобальная ф-ция).
5) generate - позволяет заполнить контейнер, например случаныеми данными. А тут числами фибоначи. Так как генератор для фибоначи не простая вещь, то нужен объект где можно хранить состояние. Можно даже не с начала начинать, а например с 34,55.

```
fib.a1_ = 34;
fib.a2_ = 55;
FibNumberGen fib;
generate( v2.begin(), v2.end(), fib);
```

6) set_intersection - несколько сложнея, здесь используется insert_iterator(ф-ция insert создаёт объект insert_iterator). Фактически этот итератор при каждом присваивание выполняет само присваивание через insert по текущему итератору, и затем увеличивает текущий итератор). Можно выполнять и над вектором, но для этого векторы нужно вначале отсортировать
7) accumulate - это из модуля <numeric> и операция plus<int> из <functional>

5. valarray и slice
Так просто чтобы знали что есть такое. valarray - массив для выполнения математических операций. Поддерживает срезы - это набор индексов. То есть при помощи среза можно сделать что-то над элементами начиная со 2-ого индекса через 5 индексов 3 раза. То есть индексы 2, 2 + 5, 2 + 2*5 = 2, 7, 12

Практика:
1) фигуры где есть for => for_each
2) посчитать кол-во фигур вверхний и нижний части экрана => count_if
3) общая площадь фигур,

Если через accumulate, думаю что надо подсказку дать, например сигнатуру ф-ции, что-то вроде

```
class Figure {
...
    static float SumArea(int acc, const Figure *f) {
        // acc – накопленный результат с предыдущих вызово
        // f – указатель на фигуру
        return 0; // нужно вернуть накопленную результат который включает в
себя данные от текущей фигуры
    }
...
}
 cout << accumulate(figures.begin(), figures.end(), 0, Figure::SumArea) <<
endl;

в результате что-то такое:
    static float SumArea(int acc, const Figure *f) {
        return acc + f->Area();
    }
```

Можно при помощи for_each
http://cplusplus.com/reference/algorithm/for_each/
Но, внимания, for_each делают копию компаратора, но возвращает аккумулированный объект. Для суммы можно юзать что-то подобное:

```
        struct AreaSummator {
            float total;
            bool operator()(const Figure *f)
            {
```

```
                total += 0; // тут нужно добавить площадь текущей финуры
            }
        };
        AreaSummator area = for_each(figures.begin(), figures.end(),
AreaSummator);
        cout << area.total;
```

---

1) Есть список студентов(Имя, Фамилия, Оценки).
Студенты - это вектор(или список) объектов Студент
Оценки - это map по <имя предмета - строка, оценка - инт>
Сделать меню:
1) Посмотреть - вывести всех (желательно в одной строке, например :
Имя Фамилия, физика 8, математика 9
2) Найти - вывести определённого студента по имени, фамилии
3) Средний балл для студента(accumulate, for_each но тогда for_each(...SomeClass).total)
4) Средний балл для предмета
5) Удалить студента по фамилии - всех студентов с совпадением под строки(remove_if)
6) Удалить предмет
...
Нельзя использовать циклы, только алгоритмы.
Захардкодить 5 студентов, с разными предметами

Внимания, for_each делают копию компаратора, но возвращает аккумулированный объект. Для суммы можно юзать что-то подобное:

```
int addMark ( int total, const map< std::string, int >::value_type& data )
{
   return total + data.second;
}
   map<string, int > m;
   m[ "a" ] = 1;
   m[ "b" ] = 2;
   const int total = std::accumulate( m.begin(), m.end(), 0, addMark);
   cout << total << endl;
```

2) Дальше можно игрушку дописывать

```
001 #include <map>
002 void TestMap()
003 {
004     typedef map<string, int> MyMap;
005     typedef map<string, int>::iterator MyMapIt;
006     typedef map<string, int>::const_iterator MyMapCIt;
007     typedef pair<string, int> MyMapEl;
008     MyMap m1;
010     m1["key1"] = 100;
011     m1["key2"] = 200;
012     for( MyMapCIt it = m1.begin(); it != m1.end();++it)
013     {
015       cout << it->first << " = " << it->second << endl;
016     }
017     cout << m1["key1"] << endl;
018     // BAD compare
019     if ( m1["key3"] == 300 )
020     {
021         cout << "key3 yes 300" << endl;
022     }
023     if ( m1["key3"] == 0 )
024     {
025         cout << "key3 yes 0" << endl;
026     }
027     if (  m1.find( "key4" ) == m1.end() )
028     {
029         cout << "key 4 not found" << endl;
030     }
031     MyMapCIt itKey1 = m1.find( "key1" );
032     if ( itKey1 != m1.end() )
033     {
034         cout << "key 1 found" << endl;
035         if ( itKey1->second == 100 )
036         {
037             cout << "and equal 100" << endl;
038         }
039     }
040     m1["key2"] = 201;
041     MyMapIt itKey2 = m1.find( "key2" );
042     if ( itKey2 != m1.end() )
043     {
044         itKey2->second = 202;
045     }
046     cout << m1["key2"] << endl;
047 }
048 /*
049 empty Test whether container is empty
051 insert Insert elements
052 erase Erase elements
053 swap Swap content
054 clear Clear content
055 find Get iterator to element
056 count Count elements with a specific key*/
059 #include <set>
060 void TestSet()
061 {
062    typedef set<string> MySet;
063    typedef MySet::iterator MySetIt;
064    typedef MySet::const_iterator MySetCIt;
067    MySet data;
068    data.insert("key1");
069    data.insert("key2");
070    data.insert("key1");
071    for(MySetCIt it=data.begin(); it!=data.end(); ++it)
072    {
073        cout << *it << endl;
074    }
075    data.erase("key2");
076    cout << data.size() << endl;
077    data.insert("key3");
078    if ( data.count("key3") != 0 )
079    {
080        cout << "found" << endl;
081    }
082    MySetIt it = data.find("key3");
083    if ( it != data.end() )
084    {
085        data.erase( it );
086        cout << data.size() << endl;
087    }
088 }
```

```
/*
090 empty Test whether container is empty
091 size Return container size
092 insert Insert element
093 erase Erase elements
094 swap Swap content
095 clear Clear content
096 find Get iterator to element
097 count Count elements with a specific value */
099
100 #include <bitset>
101 void TestBitSet()
102 {
103     enum
104     {
105         OptCanRead,
106         OptCanWrite,
107         OptCanDelete,
108         OptCanCreate,
109         OptSize
110     };
111
112     bitset<OptSize> user1;
113     user1.set( OptCanRead );
114     user1.set( OptCanWrite );
115     bitset<OptSize> user2;
116     user1.set( OptCanRead );
117     if ( user1[ OptCanRead ] )
118     {
119         cout << "you can read" << endl;
120     }
121     if ( !user2[ OptCanWrite ] )
122     {
123         cout << "you can't write" << endl;
124     }
125 }
126
127 #include <algorithm>
128 #include <numeric>
129 #include <functional>
130 bool ifMod3Func( int i )
131 {
132     return ( ( i % 3 ) == 0 );
133 }
134
135 class Data
136 {
137 friend class Printer;
138 public:
139     struct Printer
140     {
141         void operator()( const Data &d )
142         {
143             cout << d.name_ << endl;
144         }
145     };
146     static void print( const Data &d )
147     {
148         cout << d.name_ << endl;
149     }
150     Data( const string &name ) :
151         name_( name )
152     {
153     }
154
155 private:
156     string name_;
157 };
158
159 void TestAlgorithm()
160 {
161     typedef vector<int> MyVector;
162     typedef MyVector::iterator MyVectorIt;
163     int cInts[] = { 10, 20, 30 ,40 };
164     int cIntsSize = sizeof( cInts ) / sizeof( cInts[0] );
165     // Works with C-arrays
166     int *p = find( cInts, cInts + cIntsSize, 30 );
167     cout << "found " << *p << endl;
168
169     MyVector vInts( cInts, cInts + cIntsSize );
170     MyVectorIt it = find(vInts.begin(),vInts.end(),30);
212             if ( a1_ == 0 )
```

```
171     if ( it != vInts.end() )
172     {
173         cout << "found " << *it << endl;
174     }
176     it = find_if( vInts.begin(), vInts.end(),
                      ifMod3Func );
177     cout << "first mod 3 is " << *it << endl;
179     struct IfMod3 // compile with "-std=c++0x"
180     {
181         bool operator()( int i )
182         {
183             return ( ( i % 3 ) == 0 );
184         }
185     };
186     it = find_if( vInts.begin(),vInts.end(),IfMod3() );
187     cout << "first mod 3 is " << *it << endl;
188
189     MyVector v20and30(2);
190     v20and30[0] = 20;
191     v20and30[1] = 30;
192     it = search( vInts.begin(), vInts.end(),
                     v20and30.begin(), v20and30.end() );
193     cout << "20 and 30 found at " <<
                distance( vInts.begin(), it ) << endl;
195     vector< Data > data;
196     data.push_back( Data( "asd" ) );
197     data.push_back( Data( "qwe" ) );
198     for_each(data.begin(),data.end(),Data::Printer());
199     for_each( data.begin(), data.end(), Data::print );
200
201     struct FibNumberGen
202     {
203         int a1_;
204         int a2_;
205         FibNumberGen() :
206             a1_(0),
207             a2_(0)
208         {
209         }
210         int operator() ()
211         {
213             {
214                 a1_ = 1;
215                 return 1;
216             }
217             if ( a2_ == 0 )
218             {
219                 a2_ = 1;
220                 return 1;
221             }
222             int x = a1_ + a2_;
223             a1_ = a2_;
224             a2_ = x;
225             return x;
226         }
227     };
228     MyVector v2( 10 );
229     generate( v2.begin(), v2.end(), FibNumberGen());
230     for( MyVectorIt it=v2.begin(); it!=v2.end(); ++it )
231     {
232         cout << *it << endl;
233     }
234
235     set< int > s1;
236     set< int > s2;
237     set< int > s3;
238     s1.insert(10); s1.insert(20); s1.insert(30);
239     s2.insert(13); s2.insert(10); s2.insert(20);
240     set_intersection( s1.begin(), s1.end(), s2.begin(),
            s2.end(), inserter( s3, s3.begin() ) );
242     for( set< int >::const_iterator it = s3.begin(); it !
= s3.end(); ++it )
243     {
244         cout << *it << endl;
245     }
246
247     cout << accumulate( s3.begin(), s3.end(), 0,
                            plus< int >() ) << endl;
248
249 }
```

9. Потоки

1. void TestStdStream()
3 стандартных потока ввода\вывода.

2. void TestFileStreamIn
Файловый поток ввода. Можно считывать по символьно, по строчно, по элементно. .get/getline/>> возращают сам поток. Напомнить что нужно эскейпить виндовые пути к файлам. Потоки можно использовать в условиях, смысол такой - если с потоком всё хорошо(не закончился, не было ошибок), то вернёт true
Состояния потоков ios::good, ios::eof, ios::bad(Check whether badbit is set), ios::fail(Check whether either failbit or badbit is set)
badbit - i/o error, failbit - logical error
file.good()/eof()/bad()/fail()

3. void TestFileStreamOut()
Поток вывода.

4. void TestFileBinary()
У потоков есть и seekp, seekg методы если надо бегать по файлу. Фаил можно открыть как для записи так и для чтения.
<<, >> - не используются для бинарных файлов

5. void TestStringStream()
Строковый поток. можно работать со строкой как с текстовым файлом.

Практика:
сохранить в текстовый поток(Save to File/Load from File):
square 100 120 4
circle 200 100 10

+ Вот это надо обсудить на практике о полиморфизме.
Можно написать операторы >> , << для Square и Circle, но так как >> , << работают с только с теми типами для которых они определены, то для работы через базовый класс надо определить для базового.

```
class Figure
    virtual string getType() const = 0;
    virtual void toStream( ostream &os ) const = 0;
    virtual void fromStream( istream &is ) = 0;
ostream & operator << ( ostream &os, const Figure &f )
{
    f.toStream( os );
    return os;
}
istream & operator >> ( istream &is, Figure &f )
{
    f.fromStream( is );
    return is;
}
```

как записать, по всем фигурам f:

```
fs << f->GetType() << " " << *f << endl;
// Square 100 200 10
```

как прочитать:

```
string type;
while( fs >> type )
{
    Figure *f = FigureFactory::build(type);
    fs >> *f;
    ...
}
```

```
001 #include <iostream>
002
003 struct Person
004 {
005     string name_;
006     string surname_;
007 };
008 ostream & operator << ( ostream &os, const Person &p)
009 {
010     os << p.name_ << " " << p.surname_;
011     return os;
012 }
013
014 istream & operator >> ( istream &is, Person &p)
015 {
016     is >> p.name_;
017     is >> p.surname_;
018     // when error
019     // is.setstate(ios::failbit);
020     return is;
021 }
022
023 void TestStdStream()
024 {
025     cout << "stdout" << endl;
026     cerr << "stderr" << endl;
027     string s, s2;
028     cin >> s; // stdin
029
030     cout << "word1 " << "word2" << endl;
031     cin >> s >> s2;
032
033     Person p;
034     p.name_ = "John";
035     p.surname_ = "Smith";
036     cout << p << endl;
037     Person p2;
038     cin >> p2;
039     cout << p2 << endl;
040
041 }
042
043 #include <fstream>
044 void TestFileStreamIn()
045 {
046     // "d1.txt: 12345"
047     fstream f1 ("d1.txt", fstream::in);
048     if (!f1)
049     {
050         cout << "can't open d1.txt" << endl;
051         return;
052     }
053     char c;
054     while ( f1.get( c ) )
055     {
056         cout << "Get : " << c << endl;
057     }
058     f1.close();
059
060     fstream f2 ("d_lines.txt", fstream::in);
061     string line;
062     while ( getline( f2, line ) )
063     {
064         cout << "Get line: " << line << endl;
065     }
066     f2.close();
067
068     fstream f3 ("d2.txt", fstream::in);
069     int x;
070     while ( f3 >> x )
071     {
072         cout << "Get: " << x << endl;
073     }
074     f3.close();
075 }
076
077 #include <iomanip>
078 void TestFileStreamOut()
079 {
080     fstream f1 ("d_out.txt", fstream::out);
081     f1.put('A');
082     f1 << 'B' << endl;
```

```
083     string s =  "x=";
084     int x  = 123;
085     f1 << s << x << endl;
086     f1 << setprecision(2) << 1.23456 << endl;
087     f1.close();
088     fstream f2( "d_out.txt",
                    fstream::out | fstream::app );
089     f2 << "More data" << endl;
090     f2.close();
091 }
092
093 void TestFileBinary()
094 {
095     vector<int> data(10);
096     for(unsigned int i=0; i < data.size(); ++i )
097     {
098         data[i] = i*i;
099     }
100     fstream f1( "d.bin",
                    fstream::out | fstream::binary);
101     f1.write( (char *)&data[0],
                  data.size() * sizeof(int) );
102     f1.close();
103
104     fstream f2("d.bin", fstream::in | fstream::binary);
105     int x;
106     while ( f2.read( (char *)&x, sizeof(x) ) )
107     {
108         cout << x << endl;
109     }
110     f2.close();
111
113     fstream f3( "d.bin",
             fstream::in | fstream::binary | fstream::ate);
114     int fileSize = f3.tellg();
115     int intsCount = fileSize / sizeof(int);
116     vector<int> v( intsCount );
117     f3.seekg( 0, fstream::beg );
118     f3.read( (char *)&v[0], fileSize );
119     f3.close();
120     for(unsigned int i=0; i < v.size(); ++i )
121     {
122         cout << v[i] << endl;
123     }
124 }
125
126 #include <sstream>
127 void TestStringStream()
128 {
129     stringstream ss;
130     ss << 110 << " " << "120" << " " << true;
131     string s;
132     int x ;
133     bool b;
134     ss >> s >> x >> b;
135
136     cout << s << " " << x << b << endl;
137     cout << ss.str() << endl;
138 }
```

10. Операторы

В C++ почти все операторы можно перегрузить. По умолчанию оператор + работает с числами. Но например надо сложить две строки. В С это не возможно, а в C++ две string переменные можно. Например есть два векотора Vector2d , можно опередилить оператор + для сложения двух векторв.
Операторы C++ которые можно перегрузить.
arithmetic operators: + - * / % and += -= *= /= %= (all binary infix); + - (unary prefix); ++ -- (unary prefix and postfix)
bit manipulation: & | ^ << >> and &= |= ^= <<= >>= (all binary infix); ~ (unary prefix)
boolean algebra: == != < > <= >= || && (all binary infix); ! (unary prefix)
memory management: new new[] delete delete[]
implicit conversion operators
miscellany: = [] -> , (all binary infix); * & (all unary prefix) () (function call, n-ary infix)

Не стоит перегружать операторы когда смысол не очевиден. Например Вектор-Вектор в результате даст другой Вектор, но что значит string - string - не понятно.
Если перегружается оператор +, то хорошим тоном стоит перегрузить и +=, и возможно ещё и -, -=.

Операторы могут быть перегружены методами или внешними ф-циям.
Некоторые операторы всегда перегружены как методы =, [], ->, ()(ф-ция, касты)
Некоторые операторы всегда как внешние ф-ции >>, <<.

Операторы потока ввода, вывод рассмтаривались на прошлом занятии.

1. void TestAssignment()
Виды присваивания - дефолтный конструктор, конструктор с параметрами, копи-конструктор, оператор присваивания того же типа(компилятор его генерит, но если есть какое-то выделение ресурсов, то надо писать вручную. Дефолтный просто копирует все данные при помощи оператора =(shallow copy,memberwise)).
Рассказать разницу между
Class a = otherA; // calls copy constructor
a = otherA; // calls operator =
Оператор присваивания возможен с другим типом. Оператор присвивания можно выполнять цепочкой.
Напомнить о const, важный элемент проектирования класса.

2. void TestArithmetic()
Оператор+ как метод, оператор+ с другим типом(и симметричная версия), оператор- как внешняя ф-ция, оператор += меняет само содержимое объекта. Предыдущие возвращали новую копию. Для операторв у которых есть симметричная версия предпочитительноее использовать внешнию ф-цию, так как это даёт возможность написать оператор где первый аргумент это не this указатель.

3. void TestCompare()
Надо явно прописывать операции сравнения. Например == реализовать, а != можно через ! от ==. Тоже самое и с >, <

4. void TestUnary()

Можно переопределить -(унарный). Разница между постфиксной и префиксной ++. Пусть скажут результат(чему равно m3, m2)

5. void TestBrackets()
Оператор круглые скобки уже знаком с темы об алгоритма(functional object). Он может быть и с параметрами. В данном примере позволяет получить сумму в латах.
Может и с двумя параметрами, например для класса матрицы что-то подобное: double& Matrix::operator()(const int col, const int row)
Можно переопределять касты к другому типу. Но с этим надо быть осторожным. Компилятор позволяет сделать одно неявное преобразование. Данный пример позволяет направить в поток объект как double(cout << m1). Это не очевидно, поэтому лучше предпочитать метод to_double(cout << m1.to_double())

6. void TestSubscript()
Можно переопределить []. При том индекс может быть любым типом. Как например для map. Есть две версии, одна для const объекта, другая для обычного.

7. Можно даже перегружать оператор new/new[] delete/delete[] . Обычно используется для улучшения быстродействия, иди для отслеживание утечек памяти

Операторы имеют область видимости, можно например закрыть оператор = сделая его приватным.

Практика:
Простой вариант:
1) Point : =, +, +=, *, *= (то что потребуется)
Например:
оператор += для перемещения фигур
p += Point(1, 0) // увеличть p.x на 1
p += Point(0, -1) // уменьшить p.y на 1
p += Point(0, -2) * 10 // если зажата клавиша shift, то умножить каждую координату на 10, то есть уменьшить p.y на 20

2) Кнопки +- масштабировать объектый
operator * на число
Figure *f = new Figure(…);
(*f) *= 0.1; // должно увеличить фигуру на 10%
Операторы не могут быть виртуальны, то надо написать оператор для фигуры а внутри кода просто вызвать виртуальный метод.
То есть в фигуре virtual void scale(double factor) = 0;

3) Сделать новый класс Polygon – потомок от фигуры с произвольным кол-вом точек как вариант коснтруктора
Polygon(центрX, центрY, Кол-во точек)
затем перегрузить опероатор [] для установки точек. Сами координаты точек относительны точки центра
p = new Polygon(1,1, 4)
p[0] = Point(5,10); p[1] = Point(5, -10); p[2] = …; p[3] = …;

Ниже сложный(долгий) код:
Написать класс Point:
Конструктор дефолтный, с двумя параметрами x, y
Оператор =, +, - , +=, -=.
Оператор ==, !=, < (полезен для контейнеров), сравнение должно быть с точностью
Оператор *, *= (скалярное произведение ~~на вектор(DotProduct(point) (результат x1*x2 + y1*y2)),~~ на число) Методы - GetLength(), Normalize()

Написать класс Poly(имя Polygon конфликтует с WinGDI.h): !!!! НАДО НЕ БОЛЬШОЙ НАБРОСОК СДЕЛАТЬ + МОЖЕТ ПОДУМАТЬ О ПЕРЕКЛЮЧЕНИИ ФОКУСА(центр - точка, и список точек)
Оператор [] - для доступа к точкам.
Добавить метод Split(Point, Point, Polygon &newPoly) - разрезать по двум точка
Сделать редактор, меню :
сгенерить полигон(квадрат) - по центру экрана
1) масштабировать(double) - умножить
2) сдвинуть полигон на вектор - плюс, минус
3) добавить вершину между другими
- вводится номер вершины 1 (подсветить)
- вводится номер вершины 2 (подсветить)
- вводится процент 0.5(середина)
Если вводится -1 - то отмена
4) сдвинуть вершину на вектор
- вводитя номер вершины(подсветить)
- вводится вектор
5) убрать вершины стороны которых меньше заданной длины.
-вводится длина (подсветить вершины для удаления)
-подтвердить - y/n удаление

```
001 class Money
002 {
003 private:
004     double a_;
005 public:
006     Money() :
007         a_(0)
008     {}
009     Money( double santims) :
010         a_( santims )
011     {}
012     Money( double lats, double santims) :
013         a_( 100 * lats + santims )
014     {}
015     Money( const Money &m)
016     {
017         a_ = m.a_;
018     }
022     Money & operator= ( const Money &other )
023     {
024         a_ = other.a_;
025         return *this;
026     }
028     Money & operator= ( double m )
029     {
030         a_ = m;
031         return *this;
032     }
034     friend Money operator+ ( const Money &m1,
              const Money &m2 );
035     friend Money operator+ ( const Money &m1,
              double d );
036     friend Money operator+ ( double d,
              const Money &m1 );
037     friend Money operator- ( const Money &m1,
              const Money &m2 );
038     void operator+= ( const Money &other );
039
040     bool operator== ( const Money &other ) const
041     {
042         return fabs( a_ - other.a_ ) <= 0.001;
043     }
044     bool operator!= ( const Money &other ) const
045     {
046         return !( *this == other );
047     }
048     bool operator< ( const Money &other ) const
049     {
050         return ( a_ - other.a_ ) < 0.001;
051     }
052     bool operator> ( const Money &other ) const
053     {
054         return ( a_ - other.a_ ) > 0.001;
055     }
057     Money operator- () const
058     {
059         return Money( -a_ );
060     }
061     const Money& operator++() // prefix form
062     {
063         a_ += 1;
064         return *this;
065     }
066     Money operator++(int) // postfix form
067     {
068         double old = a_;
069         a_ += 1;
070         return Money( old );
071     }
073     double operator()( double factor ) const
074     {
075         return a_ * factor;
076     }
077     operator double() const
078     {
079         return a_;
080     }
081 };
083 Money operator+ ( const Money &m1, const Money &m2 )
084 {
085     return Money( m1.a_ + m2.a_ );
086 }
```

```
087 Money operator+ ( const Money &m, double d )
088 {
089     return Money( m.a_ + d );
090 }
091 Money operator+ ( double d, const Money &m )
092 {
093     return m + d;
094 }
095 Money operator- ( const Money &m1, const Money &m2 )
096 {
097     return Money( m1.a_ - m2.a_ );
098 }
099 void Money::operator+= ( const Money &other )
100 {
101     a_ += other.a_;
102 }
105 void TestAssignment()
106 {
107     Money m; // Default
108     Money m2( 2, 50 ); // Constructor with 2 args
109     Money m3(m2); // Copy constructor
110     m = m2 = m3; // Assignment
111     m = 1234;
112 }
114 void TestArithmetic()
115 {
116     Money m1( 2, 50 );
117     Money m2( 2, 40 );
118     Money m3 = m1 + m2;
119     Money m4 = m3 + 100.0;
120     Money m5 = m4 - m1;
121     m1 += m2 + m4 - m5;
122 }
124 void TestCompare()
125 {
126     Money m1( 1, 10 );
127     Money m2( m1 );
128     if ( m1 == m2 )
129     { /* ... */ }
132     if ( m1 != m2 )
133     { /* ... */ }
136     if ( m1 < m2 )
137     { /* ... */ }
140 }
141
142 void TestUnary()
143 {
144     Money m1(1, 20);
145     Money m2 = -m1;
146     ++m2;
147     Money m3( ++m2 );
148     Money m4( m2++ );
149 }
151 void TestBrackets()
152 {
153     Money m1( 12, 34 );
154     cout << m1(0.01) << " lats" << endl;
155     double s = m1; // type cast
156 }
158 class MoneyList
159 {
160 private:
161     Money money[10];
163 public:
164     Money& operator[] (int i)
165     {
166         return money[i];
167     }
168     const Money& operator[](int i) const
169     {
170         return money[i];
171     }
172 };
174 void TestSubscript()
175 {
176     MoneyList ml;
177     ++ml[0];
178     ml[1] = ml[0] + 100.0;
179     cout << ml[1](1) << endl;
180 }
```

11. RTTI, статик\дин. касты, auto_ptr

Динамическая идентификация типа данных (англ. Run-time type information, Run-time type identification, RTTI) — механизм в некоторых языках программирования, который позволяет определить тип данных переменной или объекта во время выполнения программы.

1. void TestTypeId()
В C++ у каждого типа есть свой ID, но лучше так не делать. typeid() возвращает type_info
вывод: 1011

2. void TestStaticCast()
Использование C-кастов считается плохим стилем, так как нету проверки на тип.
static_cast - конвертирование выражение в тип со статической проверкой(на этапе компиляции).
Годится как для простых типов, так и для объектов(downcast - не всегда безопасен, зато возможно сменить интерфейс на более специализированный, upcast безопасен(всегда, если не извращаться))
Статик касты могут работать и с ссылками. Спросить почему закоментированы вызовы с ссылками.

3. void TestDynamicCast()
Динамический каст отличается от статического - что он выполняется в рантайме и несколько уменьшает быстродействие. Зато есть возможность проверить тип переменной. Вернёт 0 - если что-то не так, для ссылок выбросит исключение(так как 0 невалидное значение). Об исключениях потом.
Чтобы dynamic_cast, typeid() работал нужно включить RTTI(то есть в объектном файле будет это информация). Для оптимизации его отключают и компилятор не генерит RTTI данные(VMT всё равно работает)
Для gcc 4.7 есть ключ -fno-rtti для отключения RTTI

4. void TestAutoPtr()
auto_ptr - это замена C-ишного alloc/malloc/free. Идея в конструкторе передать адрес выделенной памяти, а в деструкторе - осовободить. Так как деструктор вызывается автоматически, то освобождение тоже будет вызвано(для объектов ещё и деструктор). У auto_ptr перегружены операторы ->, *,=(можно переназначать ответственность за освобождение), release\reset
В C++11 auto_ptr депрекатед, вместо этого unique_ptr.
auto_ptr - не умеет работать с массивами(new Data[100]), unique_ptr - умеет
auto_ptr , unique_ptr - нельзя использовать в контейнерах

5. void TestSharedPtr()
shared_ptr только в C++11(или в boost). Похоже на auto_ptr - но имееют счётчик ссылок. То есть помимо указателя на сам объект, есть ещё указатель на структуру для счётчика ссылок. Когда происходит присваивания то +1 счётчик, когда укзатель выходит из области видимости то -1 счётчик. Когда счётчик = 0, то можно освободить объект так как ссылок из внешнего кода не должно быть. Поэтому не стоит мешать C-указатели и C++ умные указатели, может плохо закончиться.
shared_ptr - можно использовать в контейнерах, так как как они поддреживают операцию=(и реализованы в плане памяти просто)

6. void TestReinterCast()
reinterpret_cast аналог С-каста без каких либо проверок . В С++ принято использовать reinterpret_cast когда нужно использовать С-каст, но этот код пахнет дурно.

Если есть время, можно поговорить о std::shared_ptr и о вечных объектах в памяти из-за неправильного использования.

Практика:
1) переписать на shared_ptr, в коде не должно быть вызовов delete
2) сделать в ScreenSaver::Next:
- там есть Move\Next для фигуры через укзатель на базовый класс
- через dynamic_cast для Круга под рандом менять радиус через SetRadius, для квадрата - менять сторону через SetSide
3) + какие нибудь фишки для игры(см. лекцию 5)

___

```
001 #include <typeinfo>
002 class A
003 {
004 public:
005     // Without error when dynamic cast
006     // source type is not polymorphic
007     virtual ~A(){};
008
009     const char *AId()
010     {
011         return "A";
012     }
013 };
014 class B : public A
015 {
016     int x;
017 public:
018     const char *BId()
019     {
020         return "B";
021     }
022 };
023 class C : public B
024 {
025 public:
026     const char *CId()
027     {
028         return "C";
029     }
030 };
031 class D : public A
032 {
033 public:
034     const char *DId()
035     {
036         return "D";
037     }
038 };
039
040 typedef int myint;
041 void TestTypeId()
042 {
043     A a;
044     cout << ( typeid( A ) == typeid( a ) ) << endl;
045     B b;
046     cout << ( typeid( A ) == typeid( b ) ) << endl;
047     myint i;
048     cout << ( typeid( int ) == typeid( i ) ) << endl;
049     A *p = &b;
050     cout << ( typeid( B ) == typeid( *p ) ) << endl;
    }
051 void RefStaticCast( B &b )
052 {
053     cout << static_cast< C & >( b ).CId() << endl;
054 }
055
056 void TestStaticCast()
057 {
058     int i = 10;
059     double x = 5.5;
060     cout << static_cast< int >( x ) << endl; \\ 5
061     cout << static_cast< double >( i ) / 3 << endl;
062     //cout << ( char *)&x << endl; // will compile
063     //cout << static_cast< char * >( &x ) << endl; //
will not compile
064
065     A *b = new B();
066     B *b2 = new B();
067     A *c = new C();
068     A *d = new D();
070     // This is downcasting, it is not always safe
071     B *pb = static_cast< B * >( b );
072     cout << pb->BId() << endl;
073     pb = static_cast< B * >( c );
074     cout << pb->BId() << endl;
075     C *pc = static_cast< C * >( c );
076     cout << pc->CId() << endl;
077     //pb = static_cast< D * >( b2 ); // compile error
078
079     // THIS WILL COMPILE, BUT THIS IS ERROR CODE
080     // pb = static_cast< B * >( d );
081     // cout << pb->BId() << endl;
```

```
082
083     C c2;
084     D d2;
085     RefStaticCast( c2 );
086     //RefStaticCast( *b2 );
087     //RefStaticCast( d2 );
088     delete b; delete b2; delete c; delete d;
089 }
090
091 void PointerDynCast( A *a )
092 {
093     B *b = dynamic_cast< B * >( a );
094     if ( b )
095     {
096         cout << b->BId() << endl;
097     }
098     C *c = dynamic_cast< C * >( a );
099     if ( c )
100     {
101         cout << c->CId() << endl;
102     }
103     D *d = dynamic_cast< D * >( a );
104     if ( d )
105     {
106         cout << d->DId() << endl;
107     }
108 }
109
110 void RefDynCast( A &a )
111 {
112     B &b = dynamic_cast< B & >( a );
113     cout << b.BId() << endl;
114 }
115
116 void TestDynamicCast()
117 {
118     A *a = new A();
119     A *b = new B();
120     A *c = new C();
121     A *d = new D();
122     PointerDynCast( a );
123     PointerDynCast( b );
124     PointerDynCast( c );
125     PointerDynCast( d );
126     delete a; delete b; delete c; delete d;
127
128     A aa;
129     B bb;
130     C cc;
131     D dd;
132     //RefDynCast( aa );
133     RefDynCast( bb );
134     RefDynCast( cc );
135     //RefDynCast( dd );
136 }
137
138 struct BigData
139 {
140     int d[100000];
141 };
142
143 #include <memory>
144 void DataOut1( BigData *x )
145 {
146     cout << x->d[100] << endl;
147 }
148 void DataOut2( BigData &x )
149 {
150     cout << x.d[100] << endl;
151 }
152 void TestAutoPtr()
153 {
154     auto_ptr< BigData > a( new BigData );
155     a->d[100] = 100;
156     DataOut1( a.get() );
157     DataOut2( *a );
158
159     BigData *p = new BigData;
160     auto_ptr< BigData > a2( p );
161     delete p;
162     a2.release();
163
```

```
164     auto_ptr< A > a3( new C );
165     cout << a3->AId() << endl;
166     cout << static_cast< C * >( a3.get() )->CId() <<
endl;
167     cout << static_cast< C & >( *a3 ).CId() << endl;
168 }
169
170 struct Figure
171 {
172     Figure()
173     {
174         cout << "created at " << ( unsigned int)this <<
endl;
175     }
176     virtual ~Figure()
177     {
178         cout << "destroyed at " << ( unsigned int)this
<< endl;
179     }
180     virtual void Draw() = 0;
181 };
182 struct Circle : public Figure
183 {
184     virtual void Draw()
185     {
186         cout << "circle draw at " << ( unsigned
int)this << endl;
187     }
188 };
189 struct Square : public Figure
190 {
191     virtual void Draw()
192     {
193         cout << "square draw at " << ( unsigned
int)this << endl;
194     }
195 };
196
197 void TestSharedPtr()
198 {
199     cout << "----" << endl;
200     typedef shared_ptr< Figure > SPFigure;
201     vector< SPFigure > v;
202     for( unsigned int i = 0; i < 2; i++ )
203     {
204         v.push_back( SPFigure( new Square ) );
205         v.push_back( SPFigure( new Circle ) );
206     }
207     for( unsigned int i = 0; i < v.size(); i++ )
208     {
209         v[i]->Draw();
210     }
211     vector< SPFigure > v2;
212     v2.push_back( v[0] );
213     v2.push_back( v[1] );
214     for( unsigned int i = 0; i < v2.size(); i++ )
215     {
216         v2[i]->Draw();
217     }
218 }
219
220 void RCastC( void * p )
221 {
222     C * c = reinterpret_cast< C * >( p );
223     cout << c->CId() << endl;
224 }
225
226 void TestReinterCast()
227 {
228     int i;
229     const char *s = "1234567";
230     i = (int)s;
231     cout << (char *)i << endl;
232     i = reinterpret_cast< int >( s );
233     cout << reinterpret_cast< char *>( i ) << endl;
234
235     unique_ptr< C > c( new C );
236     RCastC( c.get() );
237     unique_ptr< D > d( new D );
238     // RCastC( d.get() ); // this is compile but bad
239 }
```

12. Исключения

Обрабо́тка исключи́тельных ситуа́ций (англ. exception handling) — механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма.
Во время выполнения программы могут возникать ситуации, когда состояние внешних данных, устройств ввода-вывода или компьютерной системы в целом делает дальнейшие вычисления в соответствии с базовым алгоритмом невозможными или бессмысленными.
Использование исключений в целях контроля ошибок повышает читаемость кода, так как позволяет отделить обработку ошибок от самого алгоритма, и облегчает программирование и использование компонентов других разработчиков(альтернатива исключениям — возврат кодов ошибок, которые вынужденно передаются по цепочке между несколькими уровнями программы, пока не доберутся до места обработки, загромождая код и снижая его понятность(куча IF-ов))
Снижается скорость работы программы, так как стоимость обработки исключения, как правило, выше стоимости обработки кода ошибки. Но эти накладные расходы вступают силу только при возникновении исключения. Поэтому, в местах программы, критичных по скорости, не рекомендуют возбуждать и обрабатывать исключения, хотя следует отметить, что в прикладном программировании случаи, когда разница в скорости обработки исключений и кодов возврата действительно существенна, очень редки.

1) void TestCatch()
Исключение - это объект, который характеризует тип исключительной ситуации. При возникновении исключения программа начинает раскручивать стэк(stack unwinding, локальне объекты будут уничтожены, деструкторы вызовутся) до тех пока не найдёт соотвествующий catch-блок. catch работает с полиморфизмом, то есть можно ловить потомков через базовый класс.
2) void TestThrow()
Можно выбрасывать и простые типа, правда особого смысла в этом нету.
ThrowB - можно выбрасывать и new A() - но тогда надо ловить A* да и потом надо освобождать объект(если уж очень нужно выбросить указатель(например 10МБайтный дамп памяти), то можно смотреть в сторону умных указателей). Поэтому проще создать временный объект и выбросить его.
Когда объект выбрасывается, то на самом деле выбрасывается копия, поэтому ThrowB2 может выбросить копию локального объекта.
3) void TestFinal()
В c++ нету finally(блок кода который выполнится всегда, вне зависимости было исключение или нет. Но есть возможность написать очень похожее на finally
(...) - поймает все исключения, throw - без аргументов выбросит текущее исключение ещё раз, может быть вызван только в блоке исключения.
При помощи (...) можно подавить все исключения(silence)
Ещё одна польза умных указателей при раскрутки стэка.

4) void TestConstructor.
Конструкторы\деструкторы и исключения.
Конструкторы и деструкторы не должны выбрасывать исключаения. Например A<-B<-C
Конструкторы вызовутся в порядке A,B,C. Что будет если в B возникнет исключение?
конструктор A сработал, конструктор C не сработал.

Деструкторы в обратно порядке C, B, A. Что будет если в B возникнет исключение? деструктора C сработал, деструктор A не сработал.
Вобщем так нельзя, а что делать если нужно, то например вынести в отдельную ф-цию Init|Done, а сами конструторы\деструткторы сделать простыми. Напомнить, что виртуальные ф-ции тоже не стоит вызывать в конструкторах\деструкторах.

5) void TestSpec()
C++11 deprecated.
В документации к библиотеке должно быть описаны исключение которые могу возникнуть. Но можно указать это и в коде.
Проверка происходит в рантайме поэтому F2() нельзя вызвать. Так как на этапе компиляции нету проверок, то специализация может повлечь к ошибкам. Вобщем-то сомнительные преимущества даёт такая специализация. В Java - есть подобное, так там проверка происходит на этапе "компиляции"(как-бы), с этим всё нормально. Но в C++ в рантайме.
throw с пустым списком говорит о том что ф-ция не выбрасывает исключения.

6) void TestScopedUsed()
От раскрутки стека можно получить вот такой утилитный класс.

7) void TestEnum()
Замена const int #define.
Начинаются с 0, и следующий +1. Можно явно указать. Внутренне это int. Именнованные енумы дают некоторую проверку на этапе компиляции на валидность значения. Но можно в енум подсунуть любой инт.
Фишка с OPT_SIZE

Практика:
Внимания во время удаление читать о "Iterator validity", для вектора и списка - erase вернёт следующий элемент.
добавить исключения:
Базовое исключение для логики, ELogic
1) Добавить игрока, при столкновении выбросить исключение EHit. Поймать - снять -1 жизнь и уничтожить объект.
2) Фигура может решить что она хочет разделится. Выбросить EDivide класс, а менеджер потом вызовет метод new_figure = figure->Divide().
3) При столкновении двух объектов выбросить исключение ECollision - добавить туда информацию об столкнувшихся объектов. Мэнеджер должен поймать и объект который более маленький должен исчезнуть.
---
1) Или так, опционально можно сделать базовый класс EApp:
EBorderCollision (с enum Border { TOP, BOTTOM, LEFT, RIGHT })
EFigureCollision (может с указателем на фигуру с которой с толкнулось)
EPlayerCollision (если с игроком)
EFigureDivide (как деление клетки, например, текущее уменьшить в два раза, и добавить ещё одну + поменять центры и скорости)

```
001 struct A
002 {
003     int x_;
004     virtual ~A(){};
005 };
006 struct B : public A
007 {
008 };
009 struct C : public A
010 {
011     void Init()
012     {
013         // ...
014     }
015     void Done()
016     {
017         // ...
018     }
019 };
021 #include <stdexcept> // out_of_range
022 #include <typeinfo> // bad_cast
023 //www.cplusplus.com/reference/exception/exception/
024 void TestCatch()
025 {
026     try
027     {
028         int *d = new int[1000000000];
029         cout << "You will not see this" << endl;
030     }
031     catch( const exception &e )
032     {
033         cout << "std::exception: " << e.what() << endl;
034     }
035     try
036     {
037         vector< int > v( 10 );
038         v.at(20) = 100;
039     }
040     catch( const out_of_range &e )
041     {
042         cout << "Out of Range: " << e.what() << endl;
043     }
044     try
045     {
046         A *b = new B();
047         C &c = dynamic_cast< C & >( *b );
048     }
049     catch( const bad_cast &e )
050     {
051         cout << "Bad cast error: " << e.what() << endl;
052     }
053     catch( const exception &e )
054     {
055         cout << "Other error: " << e.what() << endl;
056     }
057 }
058
059 void ThrowInt()
060 {
061     throw 42;
062 }
063 void ThrowCString()
064 {
065     throw "Exception!";
066 }
067 void ThrowB()
068 {
069     throw B();
070 }
071 void ThrowB2()
072 {
073     B b;
074     b.x_ = 42;
075     throw b;
076 }
077 void TestThrow()
078 {
079     try
080     {
081         ThrowInt();
082     }
083     catch ( int &x )
```

```
084     {
085         cout << "Catch " << x << endl;
086     }
087     try
088     {
089         ThrowCString();
090     }
091     catch ( const char *x )
092     {
093         cout << "Catch " << x << endl;
094     }
095     try
096     {
097         ThrowB();
098     }
099     catch ( const A &a )
100     {
101         cout << "Catch A " << a.x_ << endl;
102     }
103     try
104     {
105         ThrowB2();
106     }
107     catch ( const A &a )
108     {
109         cout << "Catch A " << a.x_ << endl;
110     }
111 }
112
113  void TestFinal()
114  {
116     int *p = new int;
117     try
118     {
120         ThrowInt();
121         delete p;
122     }
123     catch(...)
124     {
125         delete p;
126         cout << "something wrong" << endl;
127     }
129     try
130     p = new int;
131     {
132         try
133         {
134             ThrowInt();
135         }
136         catch( const A &a )
137         {
138             cout << "A" << endl;
139         }
140         delete p;
141     }
142     catch(...)
143     {
144         delete p;
145     }
147     unique_ptr< int > p2( new int );
148     try
149     {
150         ThrowInt();
151     }
152     catch( const bad_cast &e )
153     {
154         cout << "Bad cast error: " << e.what() << endl;
155     }
156     catch( const exception &e )
157     {
158         cout << "Other error: " << e.what() << endl;
159     }
160     catch(...)
161     {
162         cout << "can't deal with this" << endl;
163         // throw; //re-throw, someone else can catch it
164     }
165  }
166
167  void TestConstructor()
168  {
```

```
169      C c;
170      try
171      {
172         c.Init();
173      }
174      catch(...)
175      {
176          // ....
177      }
178      cout << c.x_ << endl;
179      try
180      {
181         c.Done();
182      }
183      catch(...)
184      {
185          // ....
186      }
187 }
189 void F1() throw( int, const char * )
191 {
192     ThrowInt();
193 }
195 void F2() throw()
196 {
197     ThrowInt();
198 }
200 void TestSpec()
201 {
202     try
203     {
204         F1();
205     }
206     catch( int )
207     {
208         cout << "int" << endl;
209     }
210     try
211     {
212         //F2(); // will abort program
213     }
214     catch( ... )
215     {
216         cout << "something" << endl;
217     }
218 }
220 class Backup
221 {
222 private:
223     char *data_;
224     unsigned int size_;
225     vector< char > copy_;
226
227 public:
228     Backup( void *data, unsigned int dataSize )
229     {
230         data_ = static_cast< char *>( data );
231         size_ = dataSize;
232         copy_.assign( data_, data_ + size_ );
233     }
234     ~Backup()
235     {
236         if ( data_ )
237         {
238             memcpy( data_, &copy_[0], size_ );
239         }
240     }
241     void Release()
242     {
243         data_ = 0;
244     }
245 };
247 void TestScopedUsed()
248 {
249     char s[100];
250     strcpy( s, "123456" );
251     cout << s << endl;
252     try
253     {
254         Backup b( s, sizeof( s ) );
255         s[3] = 'x';
256         ThrowInt();
```

```
257         b.Release();
258     }
259     catch( int )
260     {
261     }
262     cout << s << endl;
263     try
264     {
265         Backup b( s, sizeof( s ) );
266         s[3] = 'y';
267         //ThrowInt();
268         b.Release();
269     }
270     catch( int )
271     {
272     }
273     cout << s << endl;
274 }
276 enum Number
277 {
278     ZERO,
279     ONE,
280     TWO,
281     TEN = 10,
282     MINUS_ONE = -1,
283 };
285 enum Options
286 {
287     OPT1,
288     OPT2,
289     OPT3,
290     OPT4,
291     OPT_SIZE,
292 };
294 struct Data
295 {
296     enum
297     {
298         MAX_SIZE = 100,
299         MIN_SIZE = 10,
300     };
```

```
301 };
303 void FEnum( Number n )
304 {
305     cout << n << endl;
306 }
308 void TestEnum()
309 {
310     cout << TEN << endl;
311     Number x = TWO;
312     cout << x << endl;
313     // x = 3; // Invalid conversation
314     switch( x )
315     {
316         case ZERO:
317             //...
318             break;
319         case ONE:
320         case TWO:
321             //...
322             break;
323         // without default will be warning
324         // because TEN, MINUS_ONE is not handled
325         default:
326             //...
327             break;
328     }
329     cout << Data::MAX_SIZE << endl;
330     FEnum( MINUS_ONE );
331     int a = -1;
332     //FEnum( a );
333     FEnum( Number( a ) );
334     FEnum( Number( 123 ) );
335     int b = TEN; // all OK
336
337     int opt = 4;
338     if ( opt > OPT_SIZE )
339     {
340         cout << "Invalid option " << x << endl;
341     }
343 }
```

13. Шаблоны

Шабло́ны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).
В C++ возможно создание шаблонов функций и классов.
Шаблоны предоставляют краткую форму записи участка кода, но это не сокращает исполнимый код, так как для каждого набора параметров компилятор создаёт отдельный экземпляр функции или класса. В какой-то степени это похоже на очень навороченный макрос #define

1. TestFuncTemplate
Ключевое слово typename появилось сравнительно недавно, поэтому стандарт допускает использование class вместо typename. В некоторых ситуация class вводит в заблуждение компилятор. Поэтому лучше использовать typename.
Индентификаторы шаблонов - это как переменные, только в случае шаблона в этой переменной хранится тип(который обрабатывается во время компиляции, а не в рантайме)
GetMin<const char *> - можно указать явно тип для которого вызывается ф-ция, в данном случае надо, так как компилятор возьмёт непосредственный тип переменных, то есть int и double, а ф-ция GetMin работает с однотипными. int и double совместимы с double.
Пример с Compare может сравнить два разных типа.
И пояснить на всякий, что можно T - это тип, поэтому возможно
T x, T &x, T *x, const T &x, T x[10], vector< T >

2. TestFuncTemplateWithClasses
Засчёт операторов можно сделать чтобы ф-ция Avg работала с классом Точка, или даже со строкой(хотя результат странный)

3. TestClassTemplate
Шаблонные классы. Дефолтный тип для шаблона. Сам класс может быть параметризован + сам метод может и параметризация для методов. В данном случае оператор + принимает шаблон, чтобы обеспечить совместимость между Point3D<float> и Point3D<double>.
Дефолтное значение для переменной определённого(не шаблонного) типа. В этом пример на стэке создаётся массив. Когда это надо, например нужно хранить буфер на стэке(для быстроты, нету выделения памяти, прямой доступ) и буфера всегда степень двойки(поэтому копий кода будет не так уж много). Фактически Size это константа(то есть вычисляется на этапе компиляции) и компилятор может оптимизировать код(если использовать переменную для размера, то с оптимизацией сложней). И если у вас буфера всё время разного размера,то так делать не стоит(так как слишком много кода будет в объектном файле, на каждый размер свой фаил).
В шаблонах могут быть другие шаблоны. То есть надо отличать SomeType< Point3D> и SomeType<Point3D<float>>
Пример метод Add где реализация вне класса.
Point4D иногда делают typedef на тип шаблона чтобы потом можно было его вытащить или дать более удобное название.
Поговорить по поводу реализаций. Принято что всё параметризированное(ф-ции, классы, методы) пишутся в заголовочных файлах. Детали довольно таки сложные.

Если по простому, разместив в файле реализации потом будут проблемы при линковки(линковщик не найдёт нужную ф-цию). А если ф-ция описана в заголовочном файле, то всё ОК, так как компилятор скомпилит код для нужных типов. То есть реализация должна находится в области видимости в момент использования шаблона.
Данный пример работает, так как всего один фаил. Поэтому шаблонные классы и ф-ции пишите внутри заголовочного файла. Если посмотреть на STL библиотеку - то там так и есть. В ANSI/ISO(1998) стандарте есть export, но он не всеми компиляторами поддерживается + в C++11 оно убрано. Есть вариант писать так же в cpp файле(можно переименовать в tpp) и потом его включать в h-фаил.

4. TestSpecTemplate
Специализация для определённого типа - в большинстве случаев это делается для оптимизации или для совместимости с каким-то определённым типом.

// Обычно тут время уже закончилось пол часа назад =), можно перекинуть на след. занятие
5. TestCallback
Callback(делегаты) на объекты. В начале объяснить проблему, что нужен указатель this для не статичных методов. Можно сделать указатель на метод и вызывать его над каким-то объектом. Затем о std::function и std::bind(C++11). Идея что function - это лёгкий объект который внутри себя может хранить адресс на this, на метод и на аргументы(если надо их сделать заранее установленными)
Внимание callback - не будут работать полиморфно даже если виртуальны. если надо, то надо писать враппер в базовом классе внутри которого просто делегировать вызов в виртуальную ф-цию.

6. TestBinders
bind1st, bind2nd - убраны в C++11.

Практика:
Что-то на шаблоны, например класс Точка из 10-ого занятия:
Конструктор дефолтный, с двумя параметрами x, y
Оператор =, +, - , +=, -=.
Оператор ==, !=, < (полезен для контейнеров), сравнение
Оператор *, *= (скалярное произведение на число)
Методы - GetLength(), Normalize()
Переделать программу чтобы везде использовался класс точка. Посмотреть как виляет использование PointImpl <double>, PointImpl <int>.
Для этого сделать typedef PointImpl <double> Point или class Point : public PointImpl<double> {};

```
001 template< typename T >
002 T GetMin( const T &x1, const T &x2 )
003 {
004     return ( x1 < x2 ) ? x1 : x2;
005 }
006
007 template< typename T >
008 void Swap( T &x1, T &x2 )
009 {
010     T tmp = x1;
011     x1 = x2;
012     x2 = tmp;
013 }
014
015 template< typename T1, typename T2 >
016 int Compare( const T1 &x1, const T2 &x2 )
017 {
018     if ( x1 < x2 )
019     {
020         return -1;
021     }
022     else if ( x1 > x2 )
023     {
024         return 1;
025     }
026     else
027     {
028         return 0;
029     }
030 }
031
032 void TestFuncTemplate()
033 {
034     cout << GetMin( 20, 10 ) << endl;
035     double d1 = 10.0;
036     double d2 = 9.5;
037     cout << GetMin( d1, d2 ) << endl;
038     cout << GetMin<double>( 10, 9.5 ) << endl;
039     string s1 = "ABC";
040     string s2 = "ABA";
041     cout << GetMin( s1, s2 ) << endl;
042     Swap( d1, d2 );
043     cout << d1 << " " << d2 << endl;
044     Swap( s1, s2 );
045     cout << s1 << " " << s2 << endl;
046     cout << Compare( 1, 2 ) << endl;
047     cout << Compare( 1, 2.0 ) << endl;
048     cout << Compare( s1, s2 ) << endl;
049     // Will try to compile but fail cause
050     // operator < in expression s1 < 2.0 is not
defined
051     // cout << Compare( s1, 2.0 ) << endl;
052 }
053
054 template< typename T >
055 T Avg( const T &x1, const T &x2 )
056 {
057     return ( x1 + x2 ) / 2;
058 }
060 struct Point
061 {
062     double x_;
063     double y_;
064     Point( double x, double y ) :
065         x_( x ),
066         y_( y )
067     {
068     }
069     Point operator+( const Point &other ) const
070     {
071         return Point( x_ + other.x_, y_ + other.y_ );
072     }
073     Point operator/( double d ) const
074     {
075         return Point( x_ / d, y_ / d );
076     }
077 };
078 ostream & operator <<( ostream &os, const Point &p )
079 {
080     os << p.x_ << " " << p.y_ << endl;
```

```
081     return os;
082 }
083
084 string operator/ ( const string &s1, int x )
085 {
086     return s1.substr( 0, s1.length() / x );
087 }
088
089 void TestFuncTemplateWithClasses()
090 {
091     cout << Avg( 2, 5 ) << endl;
092     cout << Avg( 2.0, 5.0 ) << endl;
093     cout << Avg( Point( 1, 2 ), Point( 3, 4 ) )
             << endl;
094     cout << Avg( string( "AB" ), string( "123456") )
             << endl;
095 }
096
097 template< typename Dimension = double >
098 struct Point3D
099 {
100     Dimension x_;
101     Dimension y_;
102     Dimension z_;
103     Point3D( Dimension x, Dimension y, Dimension
z ) :
104         x_( x ),
105         y_( y ),
106         z_( z )
107     {
108     }
109     template< typename T >
110     Point3D operator+( const T &other )
111     {
112         return Point3D( x_ + other.x_, y_ + other.y_,
z_ + other.z_ );
113     }
114 };
115
116 template< typename DimensionType = double >
117 struct Point4D
118 {
119     typedef DimensionType Dimension;
120     Dimension x_;
121     Dimension y_;
122     Dimension z_;
123     Dimension t_;
124     // ...
125 };
126
127 template< typename Data, int Size = 128 >
128 struct Array
129 {
130     Data d_[ Size ];
131 };
132
133 template< template< typename > class Point >
134 struct PointStorage
135 {
136     vector< Point< double > > data_;
137
138     void Add( const Point< double > &p );
139 };
140
141 template< template< typename > class Point >
142 void PointStorage<Point>::Add
          ( const Point< double > &p )
143 {
144     data_.push_back( p );
145 };
146
147 void TestClassTemplate()
148 {
149     Point3D<> p1( 1, 2, 3);
150     Point3D< float > p2( 4, 5, 6);
151     cout << sizeof( p1 ) << endl;
152     cout << sizeof( p2 ) << endl;
153
154     Point3D<> p3 = p1 + p2;
155
156     Array< char > a1;
157     Array< char, 1024 > a2;
```

```
158     cout << sizeof( a1 ) << endl;
159     cout << sizeof( a2 ) << endl;
160
161     PointStorage< Point3D > s1;
162     PointStorage< Point4D > s2;
163     cout << sizeof( s1.data_[0] ) << endl;
164     cout << sizeof( s2.data_[0] ) << endl;
165
166     s1.Add( Point3D<double>( 0, 1, 2 ) );
167
168     Point4D<> p4;
169     Point4D<>::Dimension d1 = p4.x_;
170 }
171
172 template<>
173 void Swap< int >( int &x1, int &x2 )
174 {
175     // Will not use temporary variable
176     x1 += x2;
177     x2 = x1 - x2;
178     x1 -= x2;
179 }
180 template<>
181 void Swap< vector< int > >( vector< int > &v1,
vector< int > &v2 )
182 {
183     v1.swap( v2 );
184 }
185
186 template<>
187 struct Point4D< char >
188 {
189     // store 4 bytes in 1 integer
190     int dim;
191     // ...
192 };
193
194
195 void TestSpecTemplate()
196 {
197     int i1 = 1;
198     int i2 = 2;
199     Swap( i1, i2 );
200     cout << "Swap " << i1 << " " << i2  << endl;
201     vector< int > v1(5, 10);
202     vector< int > v2(3, 20);
203     Swap( v1, v2 );
204     Point4D< char > p;
205     cout << sizeof( p ) << endl;
206 }
207
208 struct A
209 {
210      int F1( double d )
211      {
212          cout << "F1" << d << endl;
213          return d;
214      }
215      int F2( double d )
216      {
217          cout << "F2" << d << endl;
218          return d;
219      }
220      int F3( double d, const string &s )
221      {
222          cout << "F3" << d << s << endl;
223          return d;
224      }
225 };
226 typedef int (A::* FCallback)( double d );
227
228 #include <functional>
229
230 typedef std::function< int( double d ) > StdCallback;
231 void Caller( StdCallback &c )
232 {
233     c( 50 );
234 }
235
236 void TestCallback()
237 {
```

```
238     // pointer to member functions
239     FCallback c1 = &A::F1;
240     FCallback c2 = &A::F2;
241     A a;
242     int x1 = (a.*c1)( 10 );
243     int x2 = (a.*c2)( 20 );
244
245     // std::function
246     function< int( A&, double ) > c3 = &A::F1;
247     int x3 = c3( a, 30 );
248
249     // std::bind std::placeholders
250     function< int( double d ) > c4 =
            bind( &A::F1, &a, placeholders::_1 );
251     function< int( double d ) > c5 =
            bind( &A::F2, &a, placeholders::_1 );
252     c4(40);
253     Caller( c4 );
254     Caller( c5 );
255
256     function< int( A&, double d, const string &s ) >
c6 = &A::F3;
257     c6( a, 60, string( "aaa" ) );
258
259     function<int( double d, const string &s )> c7 =
            bind( &A::F3, &a,
                  placeholders::_1, placeholders::_2 );
260     c7( 70, string( "aaa" ) );
261
262     string s2 = "bbb";
263     function< int( double d ) > c8 =
            bind( &A::F3, &a, placeholders::_1, s2 );
264     c8( 80 );
265 }
266
267 //template <class T> struct equal_to :
                binary_function <T,T,bool> {
268 //  bool operator() (const T& x, const T& y) const
        {return x==y;}
269 //};
270
```

```
271 template <class T>
272 struct similar_to : binary_function< T, T, bool >  {
273   bool operator() (const T& x, const T& y) const
274   {
275       return abs(x-y) <= 10;
276   }
277 };
278
279 void TestBinders()
280 {
281     int a1[] = {10, 20, 32, 20, 20};
282     int a2[] = {10, 20, 40, 80, 160};
283     std::pair< int *, int * > r =
284         mismatch(a1, a1 + 5, a2, equal_to<int>() );
285     cout << "mismatch " << *r.first <<  " " <<
                *r.second << endl;
286     r = mismatch(a1, a1 + 5, a2, similar_to<int>());
287     cout << "mismatch " << *r.first <<  " " <<
                *r.second << endl;
288
289     int c1 = count_if( a1, a1 + 5,
            bind1st( equal_to<int>(), 20 ) );
290     int c2 = count_if( a1, a1 + 5, bind(
            equal_to<int>(), 20, placeholders::_1 ) );
291     int c3 = count_if( a1, a1 + 5, bind(
            similar_to<int>(), 20, placeholders::_1 ) );
292     cout << c1 << " " << c2 << " " << c3 <<
             " elements is 20" << endl;
293
294 }
```

14. C++11

Не забыть про callback с прошлой лекции(если не было рассказано)
C++11 - новая версия стандарта языка C++
В процессе работы над новым стандартом носил условное наименование C++0x.
http://www.stroustrup.com/C++11FAQ.html

1. TestConstExpr
Можно явно указать что ф-ция возвращает константное выражение на этапе компиляции, это не всегда детектить компилятор. Есть довольно таки жёсткие ограничения, по сути это однострочное константное выражение(в выражение могут вызывтся и другие ф-ции)

2. TestInitializerList
Это std::initializer_list, Не путать со списком инициализации. initializer_list - это лёгкий класс который даёт возможность инициализировать что-то массивом других объектов определённого типа. Контейнеры могут принимать списки инициализации. Можно написать свои классы которые работают со списками инициализации.
Uniform initialization для объект, но это лучше не использовать на большом отдалении определения класса от его использования, так как привязано к порядку определения полей.

3 TestAuto
Автоматическое выведение типов. Позваляет явно не указывать тип, он будет автоматически определён. typeid(variable).name - выведет его значение(не в C++ синтаксисе, но что-то подобное)
decltype - вывод типа из выражения

4.TestRangeFor
Цикл по коллекции(range-based for), использует begin(), end().

5. TestLambda
Лямбда-функции(анонимные функции). в [] указывается замыкание(closure) для ф-ции, фактически то-что будет доступно внутри ф-ции(по умолчанию локальные переменные внешней ф-ции не доступны)
Ещё в C++11 можно иницаилизировать не статик данные сразу при определение.
Для классов нужно передать указатель this чтобы иметь доступ к экземпляру. В этом примере total - передать нельзя, так как total принадлежит объекту. Замыканию нужен абсолютный адрес.
[] - пустое замыкание, как обычная ф-ция
[&] - все переменные захватываются по ссылке
[=] - все переменные захвыватываются по значению
[&, this] - this - по значение, остальные по ссылке.

6. TestClass
Override - как в Джаве. Защита от случайного создания второй виртуально ф-ции.
Вызов конструктора из другого конструктора.
В C++11 для нулевого указателя нужно использовать nullptr. Он даёт более надёжную проверку типов.
default - явно говорит что нужен дефолтный конструктор который генерит комприлятор, и delete для конструкторов и методов(вместо закрытия через приватную секцию)
final классы и методы

Практика:
Заменить for на range-based for.
Добавить в Менеджер фигур конструктор который принимает список инициализатор с Фигурами.
Добавить override для виртуальных ф-ций, которые переопределены в потомках.
Если есть синглтон, то добавить delete для копи конструктора и оператора =.
Если пользователь нажимает S(или другую) показать статистику = Кол-во фигур в квадрантах, использовать for_each + лямбду для подсчёта фигур, что-то типа такого:

```
vector<int> Manager::CalcFigures()
{
	// s1 = count_if(...
	// ...
	return { s1, s2, s3, s4 };
}
```

```
010
011 constexpr int F1( int x )
012 {
013     return 2 * x;
014 }
015 const int X1 = F1( 10 );
016
017 constexpr int valueSize()
018 {
019     return 5 + F1(2);
020 }
021
022 void TestConstExpr()
023 {
024     int value[ valueSize() ];
025     cout << sizeof( value ) << endl;
026 }
027
028 vector< int > F2()
029 {
030     return { 1, 2, 3 };
031 }
032
033 struct Data
034 {
035     vector< int > d;
036     Data( initializer_list<int> init )
037     {
038         for( const int *i = init.begin(); i !=
init.end(); ++i )
039         {
040             d.push_back( *i );
041         }
042     }
043
044     void Append( initializer_list<int> l )
045     {
046         d.insert( d.end(), l.begin(), l.end() );
047     }
048 };
049
```

```
050 void TestInitializerList()
051 {
052     vector< int > v = { 1, 2, 3, 4 };
053     cout << v.size() <<  " " << v.back() << endl;
054     map< string, int > m = { {"word1", 1}, {"word2",
2} };
055     cout << m.size() << m.begin()->first << ":"
          << m.begin()->second << endl;
056
057     vector<int> v2 = F2();
058     cout << v2[0] << v2[1] << v2[3] << endl;
059
060     Data d = { 1, 2, 3 };
061     d.Append( { 4, 5, 6 } );
062     cout << d.d.size() << endl;
063     vector< Data > d2 = { Data{1,2,3}, Data{1, 2} };
064     cout << d2.size() << d2[0].d.size() << endl;
065
066     vector< pair<string, string> > people =
067         { {"Alex", "Eve"}, {"Bob", "Ada"} };
068     cout << people[0].first << " and "
          << people[0].second << endl;
069     vector< vector<int> > v3 =
         { { 1, 2, 3 }, {4, 5 } };
070     cout << v3.size() << v3[0].size()
          << v3[1].size() << endl;
071
072     struct S1 {
073         int f1;
074         float f2;
075         string f3;
076     };
077     // Uniform initialization
078     S1 s1 = { 1, 2.0, "test" };
079     cout << s1.f3 << endl;
080     S1 s2 = { 1, 2.0 };
081     cout << s2.f3 << endl;
082
083 }
084
```

```
085 int F3( double d, const string &s )
086 {
087     cout << d << s << endl;
088 }
089
090 void TestAuto()
091 {
092     int x = 3;
093     auto x2 = 3;
094     cout << typeid( x ).name() << " "
             << typeid( x2 ).name() << endl;
095     list< int > l1 = {1, 2, 3};
096     for( auto it = l1.begin(); it!=l1.end(); ++it )
097     {
098         *it *= 2;
099     }
100     for( auto it = l1.cbegin();it!=l1.cend(); ++it )
101     {
102         cout << *it << endl;
103     }
104
105     //function<int( double d,const string &s )> f =
106     // bind(&F3,placeholders::_1,placeholders::_2 );
107     auto f = bind( &F3,
              placeholders::_1, placeholders::_2 );
108     f( 70, "aa" );
109
110     int i0 = 1;
111     decltype( i0 ) i1 = 3;
112     decltype( F3(1, "") ) i2 = 4;
113     cout << i1 << " " << i2 << endl;
114
115     vector<int> v = { 1, 2, 3 };
116     decltype( v[0] ) i3 = v[0];
117     i3 = 4;
118     cout << v[0] << endl;
119
120     decltype( (i1) ) i4 = i1;
121     i4 = 5;
122     cout << i1 << endl;
123 }
```

```
125 void TestRangeFor()
126 {
127     list< int > l1 = {1, 2, 3};
128     for( auto &x : l1 )
129     {
130         x *= 2;
131     }
132     for( const auto &x : l1 )
133     {
134         cout << typeid(x).name() << x << endl;
135     }
136 }
137
138 struct Data3
139 {
140     vector<int> v={1, 2, 3};
141     int total = 0;
142     void Foo()
143     {
144         for_each(v.begin(), v.end(), [this] (int x)
            {
145           this->total += x;
146         });
147     }
148 };
149
150 void TestLambda()
151 {
152     vector<int> v{ 10, -5, 2, -6};
153     sort( v.begin(), v.end(), [](int a, int b)
154         { return abs(a)<abs(b); } );
155     for( auto &x : v )
156     {
157         cout << x << endl;
158     }
159     int total = 0;
160     for_each( v.begin(), v.end(), [&total] (int x)
        {
161         total += x;
162     });
163     cout << total << endl;
```

```
164     auto f = find_if( v.begin(), v.end(), [=](int x)
165         { return x > total; } );
166     cout << *f << " found" << endl;
167
168     Data3 d;
169     d.Foo();
170     cout << d.total << endl;
171 }
172
173 struct Base {
174     virtual void Func(float);
175 };
176
177 struct Derived : Base {
178     // this will give error, because Func(int)
179     // creates second virtual function
180     // virtual void Func(int) override;
181
182     virtual void Func(float) override;
183 };
184
185 class SomeType  {
186     int number;
187
188 public:
189     SomeType(int new_number) : number(new_number) {}
190     SomeType() : SomeType(42) {}
191 };
192
193 struct NonCopyable {
194     NonCopyable() = default;
195     NonCopyable(const NonCopyable&) = delete;
196     NonCopyable & operator=(const NonCopyable&) =
delete;
197     int Foo() { return 42; }
198 };
199 class X : public NonCopyable
200 {
201     int Foo() = delete;
202 };
203
```

```
204 struct Base1 final { };
205 //struct Derived1 : Base1 { };
206 struct Base2 {
207     virtual void f() final;
208 };
209
210 struct Derived2 : Base2 {
211     //virtual void f();
212 };
213
214 void TestClass()
215 {
216     char *pc = nullptr;     // OK
217     int  *pi = nullptr;     // OK
218     bool   b = nullptr;     // OK. b is false.
219     // int    i = nullptr;     // error
220
221     X x;
222     X x2;
223     //x = x2;
224     //x.Foo();
225 }
```

15. Library, C++11(2), можно расширить примерс move-конструктром

1. TestLib() (это лучше сразу на практике)
Shared library: под виндовс DLL, под Unix - .lib, .so.
.a - фаил с таблицой импорта(dllimport, есть программа dlltool которая делаю таблицу из DLL). Чтобы всё это работало компилятору нужные загаловочные файлы, линковщику нужен *.a, а при запуске в системе должна быть видна DLL(либо в %PATH%, либо в директории программы).
В codeblock есть шаблон нового проекта, shared library/static library
Рекомендации по организации:
1) include - папка для всех хедоров(могут быть подпапки если надо)
Для codeblock надо прописать в Search Directories
2) lib - папка для библиотек
Сюда можно *.a закинуть
3) Для девелопмента положить DLL в bin\debug\ рядом с your_project.exe
В codeblock есть шаблон "shared library" - с настройками проект под библиотеку.

2. TestExternC
По дефолту имена ф-ций имеют C++ соглашение по декорированию имён(name mangling), это нужно чтобы обеспечить уникальное имя для ф-ции(например чтобы поддерживать одинаковые имена методов в разных классах, пространство имён). C-программы с таким декорированием не умеют работать поэтому если нужно обеспечить совместимость то применяется ключевое слов extern "C", должно быть указано в загаловке ф-ции. Идея что заголовок совместим с C, а код ф-ции на C++(то есть внутри можно использовать объекты, главное чтобы загловок был C-совместимый)

3. std::move
вызвоет move конструктор с rvalue ссылкой для реализации copy and swap idiom. Это хорошо работает с временными объектами, так как их содержимое всё равно будет утерено. std библиотека переписана и у контейнеров есть move конструкторы + некоторые ф-ции(например оператор=, push_back) имеют версию с && для временных переменных

4. TestBaseClassCall
вызов метода базового класса Base::SomeMethod

5. TestExplicit
Отключение неявного преобразования, это часто помогает избежать глупых ошибок.

Практика:
Показать как сделать шаред библиотеку:
1) new project - Shared Library, имя проекта TestLib
2) testlib.hpp:
```
void hello();
```
testlib.cpp
```
#include <iostream>
#include "testlib.hpp"
void hello() {
  std::cout << "Hello";
}
```

3) compile -> *.dll, *.a
4) new project - Console application, имя проекта RunLib

```
#include "testlib.hpp"
in main ... call hello();
```

compile, will get compilation error
5) copy testlib.hpp to project root
compile, will get linker reference error
6) copy testlib.a to lib/testlib.a and add build options->linker settings
compile, will get runtime error(library is not found)
7) copy testlib.dll to bin\debug
run => success

Задание
1) AllegroBase вынести в библиотеку.
3) Добавить move-конструктор в Vector, Figure

```
001 #include "SomeData.h"
002
003 void TestLib()
004 {
005     SomeData d1(10);
006     d1.Print();
007     SomeData d2 = CreateSomeData();
008     d2.Print();
009 }
010
011 extern "C" void Foo( int x )
012 {
013     cout << "It is foo " << x << endl;
014 }
015
016 extern "C"
017 {
018
019 void Bar1( int x )
020 {
021     cout << "It is bar1 " << x << endl;
022 }
023
024 void Bar2( int x )
025 {
026     cout << "It is bar2 " << x << endl;
027 }
028
029 }
030
031 void TestExternC()
032 {
033     Foo( 10 );
034     Bar1( 20 );
035     Bar2( 30 );
036 }
037
038 struct Obj
039 {
040     char *big;
041     Obj()
042     {
043         cout << "Created at " << (unsigned int)this
                 << endl;
044         big = new char[1000];
045     }
046     Obj( const Obj &other )
047     {
048         cout << "Created at " << (unsigned int)this
                 << endl;
049         big = new char[1000];
050         memcpy( big, other.big, 1000 );
051     }
052     Obj( Obj &&other )
053     {
054         cout << "Move at " << (unsigned int)this
                 << endl;
055         big = other.big;
056         other.big = nullptr;
057     }
058     ~Obj()
059     {
060         cout << "Destroyed at " << (unsigned int)this
                 << endl;
061         delete[] big;
062     }
063 };
064
065 #include <utility>
066 void TestMove()
067 {
068     {
069         Obj x;
070         vector< Obj > v;
071         v.reserve(2);
072         v.push_back( x );
073         v.push_back( Obj() );
074         //Output if no Move constructor exists:
075         //Created[def] at 2686696
076         //Created[copy] at 9056088
077         //Created[def] at 2686708
078         //Created[copy] at 9056092
```

```
079         //Destroyed at 2686708
080         //Destroyed at 9056088
081         //Destroyed at 9056092
082         //Destroyed at 2686696
083     }
084     string s1 = "SOME TEXT";
085     string s2 = move( s1 );
086     cout << s1 << "," << s2 << endl;
087     {
088         Obj x;
089         vector< Obj > v;
090         v.reserve(2);
091         v.push_back( move( x ) );
092         v.push_back( Obj() );
093         //Created[def] at 2686680
093.1       //Move at 7679832
093.2       //Created[def] at 2686716
093.4       //Move at 7679836
095         //Destroyed at 2686716
096         //Destroyed at 7679832
097         //Destroyed at 7679836
098         //Destroyed at 2686680
101     }
102 }
103
104 struct Base
105 {
106     int F1()
107     {
108         return 1;
109     }
110 };
111
112 struct Child : public Base
113 {
114     int F1()
115     {
116         return 2;
117     }
118     void Process()
119     {
120         cout << F1() << endl;
121         cout << Base::F1() << endl;
122     }
123 };
124
126 void TestBaseClassCall()
127 {
128     Child c;
129     c.Process();
130 }
131
132 struct Spacer
133 {
134     string s_;
135     explicit Spacer( int i ) :
136         s_( i, '_' )
137     {
138     }
139     const string & str() const
140     {
141         return s_;
142     }
143 };
145 void Print( int x, int y, const Spacer &sp )
146 {
147     cout << x << sp.str() << y << endl;
148 }
149
150 void TestExplicit()
151 {
152     Spacer s(4);
153     cout << s.str() << endl;
154     // This is not obvious
155     // Print( 1, 2, 3 ); // print '1___2'
156     Print( 1, 2, Spacer( 3 ) );
157     // This is not obvious
158     // s = 2;
159     // s = 'a';
160     //en.wikibooks.org/wiki/More_C%2B%2B_Idioms/Safe_bool
161 }
```

16. Debug output, Unittest

1. TestInline
inline ф-ции - раскрытие вызова ф-ции, это может повысить быстродействие(нету стека для передачи аргументов, нету вызова ф-ции, нету возврата результата), но код становится больше. Компилятор не всё может заинлайнить, поэтому это обычно очень простые ф-ции + при оптимизации это может либо сделать сам компилятор, либо компилятор может и не захотеть заинлайнить.
inline ф-ции должны лежать в загаловочном файле, если они используется в разных cpp файлах(если это ф-ция локальная, то можно в cpp описать реализацию).

2. TestDefault
Параметры по умолчанию. Работает и для объектов

3. TestConstCast
Снятие константного модификатора - плохая практика, но иногда из-за это надо(например из-за плохого дизайна внешней библиотеки). Можно поменять реальную константу(та что в секции для констант хранится). Применять очень осторожно(пример с интегером, компилятор сохранил значение в регистре и не перечитал из адреса)

4. TestMutable
Mutable модификтор позволяет менять переменную в константном объекте. Иногда такое надо, например для мутекса, для кэширования.

5. Logger
Макрос в C++ можно применять и так. Идея - на этапе компилиции можно объявить макрос либо пустым, либо что-то делающим. Здесь используется MAKE_DEBUG проверка(задаётся в опция компилятора как define). для дебаг версии включаем MAKE_DEBUG, для релиз этот дефайн не задаём. В итоге релиз верся работает быстро и без лишний отладочной информации - такое возможно благодаря отключения отладочного вывода на этапе компиляции. Например в джаве так нельзя сделать, то есть нельзя полностью убрать оверхеад на логирование.

6. Google unit test (1.6.0)
Юнит тесты - узконаправленное тестирование, позволяет проверить на корректность отдельные модули исходного кода программы.
Есть ещё регрессионные, функциональные - где тесттируется взаимодействие между модулями.
Пример очень простой, можно поговорить о фикстура(базовые данные для тестирование(например, база данных с тестовыми данными)), для гугл - это объект потомок от testing::Test. virtual void SetUp/TearDown,
Чтобы с билдить:
\gtest-1.6.0>g++ -I ./include -I . -c ./src/gtest-all.cc
\gtest-1.6.0>ar -rv libgtest.a gtest-all.o

На данный момент в реализации тестах используются не безопасные ф-ции(например stricmp без проверки на выход за границу буффера), в gcc с ключом -std=c++11 их использовать нельзя, но если надо можно отключить дефайн __STRICT_ANSI__

Практика:
0) Попробывать качнуть google unit test + собрать через gcc

- download and extract google unit test
- cd to folder, run g++, ar
- new console application, copy lib to lib\ folder, copy include folder
-build options linker settings + search directories for include
- make new target Test = right click project properties, add build options defines MAKE_TEST
- code :

```
int foo(x){ return x + 2;};
TEST(....) TEST_EQ(2, foo(0))
#ifdef MAKE_TEST
  testing::InitGoogle....
#else
  Run app
#end
```

1) Добавить юнит тесты(либо на утилитный класс, если его нету то на Figure или точку)

```
p1 = 1;2
p2  = 2;3
p3 = p1 + p2;
EXPECT_EQ(3, p.x)
EXPECT_EQ(5, p.y)
```

2) добавить логи в программу с фигурами(например создание и уничтожение фигур). Проверить отключение\включение логирования через define. Codeblocks подсветка тупит с макросами если через Build Options->Compiler->Defines#, можно её отключить, в Editor->General->C/C++. Сделать два билда debug, release. Должны запускатся из эксплорера.
3) Улучшить логгер - добавить возможность вывода в фаил. Лучше каждый раз не переоткрывать фаил, а использовать статик переменную для файлового потока.

```
015 #include <ctime>
016 struct Timer
017 {
018     clock_t begin_ = clock();
019     ~Timer()
020     {
021         double end = clock();
022         double secs = double( end - begin_ ) /
CLOCKS_PER_SEC;
023         cout << "Elapsed " << secs << " s" << endl;
024     }
025 };
026 int F1(int y)
027 {
028     return 42 * y;
029 }
030 inline int F2(int y)
031 {
032     return 42 * y;
033 }
034
035 struct S1
036 {
037     int F1(int y)
038     {
039         return 42 * y;
040     };
041     int F2(int y);
042     int F3(int y);
043 };
044
045 int S1::F2(int y)
046 {
047     return 42 * y;
048 }
049
050 inline int S1::F3(int y)
051 {
052     return 42 * y;
053 }
054
```

```
055 #define MEASURE_S { Timer t;
056 #define MEASURE( n ) \
057   for( long long int i = 0; i < n; ++i ) {
058 #define MEASURE_E } }
059 void TestInline()
060 {
061     MEASURE_S
062         MEASURE( 1000000000 )
063         volatile int y = F1(2);
064     MEASURE_E
065     MEASURE_S
066         MEASURE( 1000000000 )
067         volatile int y = F2(2);
068     MEASURE_E
069     MEASURE_S
070         S1 s;
071         MEASURE( 1000000000 )
072         volatile int y = s.F1(2);
073     MEASURE_E
074     MEASURE_S
075         S1 s;
076         MEASURE( 1000000000 )
077         volatile int y = s.F2(2);
078     MEASURE_E
079     MEASURE_S
080         S1 s;
081         MEASURE( 1000000000 )
082         volatile int y = s.F3(2);
083     MEASURE_E
084 //Elapsed 8.604 s
085 //Elapsed 8.593 s
086 //Elapsed 11.19 s
087 //Elapsed 11.368 s
088 //Elapsed 11.251 s
089 }
091 void F4( int x, int y = 1 )
092 {
093     cout << x * y << endl;
094 }
095 void F5( const char *s1 = "aaa", const string &s2 =
"bbb" )
```

```
096 {
097     cout << s1 << s2 << endl;
098 }
099 struct S2
100 {
101     int x_;
102     S2( int x = 2 ) :
103         x_( x )
104     {
105     }
106     void F1( int y = 0 )
107     {
108         cout << x_ + y << endl;
109     }
110 };
111 void F6( S2 s = S2( 3 ) )
112 {
113     s.F1();
114 }
115 void TestDefault()
116 {
117     F4( 2 );
118     F4( 2, 10 );
119     F5();
120     F5( "XXX" );
121     F5( "XXX", "YYY" );
122     S2 x;
123     x.F1();
124     S2 y( 10 );
125     y.F1();
126     F6();
127     F6( S2( 20 ) );
128 }
130 void ChangeS( const string &s )
131 {
132     const_cast< string &>( s ) = "BBB";
133 }
134 void PrintX( const int &x )
135 {
136     cout << x << endl;
137 }
```

```
138 void TestConstCast()
139 {
140     const string s = "AAA";
141     ChangeS( s );
142     cout << s << endl;
143
144     const int x = 10;
145     cout << x << endl;
146     const_cast< int & >( x ) = 100;
147     cout << x << endl; // still print 10
148     PrintX( x ); // print 100
149 }
150
151 struct Data
152 {
153     vector< string > v_;
154     mutable int cacheIndex_ = -1;
155     mutable string cacheKey_;
156
157     int Find( const string &key ) const
158     {
159         if ( key == cacheKey_ )
160         {
161             cout << "cache hit" << endl;
162             return cacheIndex_;
163         }
164         int i = -1;
165         for( auto x : v_ )
166         {
167             ++i;
168             if ( x == key )
169                 break;
170         }
171         if ( i >= v_.size() )
172             i = -1;
173         cacheIndex_ = i;
174         cacheKey_ = key;
175         return i;
176     }
177
178 };
```

```
179
180 void TestMutable()
181 {
182     Data d;
183     d.v_ = { "aa", "bb", "cc" };
184     cout << d.Find( "bb" ) << endl;
185     cout << d.Find( "bb" ) << endl;
186     cout << d.Find( "xx" ) << endl;
187     cout << d.Find( "xx" ) << endl;
188 }
189
190
191 #if defined( MAKE_DEBUG )
192 #define LOG( level, message ) \
193   { Logger( Logger::Log ## level, __FILE__, __LINE__,
__FUNCTION__ ) message; }
194 #else
195 #define LOG( level, message )
196 #endif
197
198 #define LOG_INFO( message ) LOG( Info, message )
199 #define LOG_WARNING( message ) LOG( Warning, message )
200 #define LOG_ERROR( message ) LOG( Error, message )
201
202 struct Logger
203 {
204     enum Level
205     {
206         LogInfo,
207         LogWarning,
208         LogError,
209     };
210     static const char *LevelNames[];
211     Logger( Level level, const char *file, int line,
const char* func ) :
212         level_( level ),
213         file_( file ),
214         line_( line ),
215         func_( func )
216     {
217         cout << LevelNames[ level_ ];
218     }
219     ~Logger()
220     {
221         cout << " (" << file_ << " " << func_ << ":"
<< line_ << ")" << endl;
222     }
223
224     template< typename T >
225     ostream & operator << ( const T &t )
226     {
227         return cout << t;
228     }
229 private:
230     Level level_;
231     const char *file_;
232     int line_;
233     const char *func_;
234 };
235
236 const char *Logger::LevelNames[] =
237 {
238     "INFO : ",
239     "WARN : ",
240     "ERROR: ",
241 };
242
243 struct S4
244 {
245     S4()
246     {
247         LOG_INFO( << "Created" );
248     }
249     ~S4()
250     {
251         LOG_INFO( << "Destroyed" );
252     }
253 };
254
255 void TestLog()
256 {
257     LOG_INFO( << "This is info" );
```

```
258     // INFO : This is info (C:\test\main.cpp
TestLog:244)
259     LOG_WARNING( << "This " << "is " << "warning" );
260     LOG_ERROR( << "This " << "is " << "error at " <<
clock() );
261     S4();
262 }
263
264 int Factorial( int n )
265 {
266     int r = 1;
267     for( int i = 2; i <= n; ++i )
268     {
269         r *= i;
270     }
271     return r;
272 }
273
274 /*
275 #ifdef __STRICT_ANSI__
276 #undef __STRICT_ANSI__
277 #endif
278 */
279 #include "gtest/gtest.h"
280 TEST(TestGroup, TestName)
281 {
282   EXPECT_EQ( 1, Factorial( 1 ) );
283   EXPECT_EQ( 2, Factorial( 2 ) );
284   EXPECT_EQ( 24, Factorial( 4 ) );
285   EXPECT_EQ( 100, Factorial( 5 ) ) << "This should
fail";
286 }
287 /*
288 [==========] Running 1 test from 1 test case.
289 [----------] Global test environment set-up.
290 [----------] 1 test from TestGroup
291 [ RUN      ] TestGroup.TestName
292 C:\VCode\Progmeistars\CPP\16\Classes\main.cpp:288:
Failure
293 Value of: Factorial( 5 )
294   Actual: 120
295 Expected: 100
296 This should fail
297 [  FAILED  ] TestGroup.TestName (5 ms)
298 [----------] 1 test from TestGroup (6 ms total)
299
300 [----------] Global test environment tear-down
301 [==========] 1 test from 1 test case ran. (11 ms
total)
302 [  PASSED  ] 0 tests.
303 [  FAILED  ] 1 test, listed below:
304 [  FAILED  ] TestGroup.TestName
305
306  1 FAILED TEST
307 */
308 TEST(Vector, Size)
309 {
310     vector< int > v;
311     EXPECT_EQ( 0, v.size() ) << "Empty vector";
312     v = { 1, 2, 3 };
313     EXPECT_EQ( 3, v.size() ) << "Assign 3 elements";
314     v.insert( v.begin(), { 4, 5 } );
315     EXPECT_EQ( 5, v.size() ) << "Append 2 elements";
316 }
317
318 TEST(Vector, Insert)
319 {
320     vector< int > v = { 1, 2, 3 };
321     EXPECT_EQ( (vector<int>{ 1, 2, 3 }), v ) <<
"Create 3 elements";
322     v.insert( v.begin(), { -2, -1 } );
323     EXPECT_EQ( (vector<int>{ -2, -1, 1, 2, 3 }), v )
<< "at begin";
324     v.insert( v.end(), { 4, 5 } );
325     EXPECT_EQ( (vector<int>{ -2, -1, 1, 2, 3, 4,
5 }), v ) << "at end";
326     v.insert( v.begin() + 3, { 7, 8 } );
327     EXPECT_EQ( (vector<int>{ -2, -1, 1, 7, 8, 2, 3,
4, 5 }), v ) << "at end";
328 }
```

1. Повтор C
2. наследование, секции защиты данных.
   + простой пример наследование фигур, allegro для графики
3. Виртуальные методы, указатель на класс, важность виртуальных деструкторов. динамические объекты, operator new\delete и версии с []
4. не забыть о ссылках рассказать, разделения на файлы cpp/hpp(#pragma некоторые директивы) . интерфейсы и абстрактные методы, множественное наследование, виртуальное наследование.
5. классы со static методами, данными. Factory, Singleton, friend классы.
6. namespace ,string, манипуляторы cout
7. стандартные контейнеры : vector, list, set , итераторы.
8. map, set, <algorithm> к контейнерам
9. потоки cout, cin, cerr, sstream, файлы
10. переопределение операторов(как сделать чтобы cout << принимал ваш тип), функциональный объект(function object), копи-конструктorhttp://stackoverflow.com/questions/4421706/operator-overloading
11. RTTI <typeinfo>, касты static, dynamic, reinterpret <memory> auto_ptr, shared_ptr(c++11).
12. enum, исключения <exception>. static const int, при дин. касте ссылок. exception и auto_ptr
13. шаблоны. Например для множеств. callback на методы(делегаты)
14. <regex>, C++11 - новые возможности, новый ф-ции у стандартных контейнеров

15. (C++11) <mutex>, <thread>(C++11)
16. **Unitest**(например при помощью googletest), debug traces с помощью макросов, assert , Static assertions

Можно попробывать написать свой контейнер, когда говорить будем о шаблонах.
Полностью это сложно, ну например для алгоритма find, sort
C++11 - фишки
Под виндой нету POSIX - поэтому треды, мутекс - не работают, а рег. выражения - глючат.
в C++11 есть поддержка utf8, utf16, utf32 - но по быстрому не понял как вывести из utf8 на консоль
Strongly typed enumerations
Variadic templates
New string literals